Wzmgr: v1.01 :: lrybicki@mat.uni.torun.pl -- wersja 0.1 z 27/5/2005r.

**UNIWERSYTET MIKOŁAJA KOPERNIKA – WYDZIAŁ MATEMATYKI I INFORMATYKI**

**Leszek Rybicki**
nr albumu: 146-975

# Modelowanie złożonych kolonii agentów z użyciem maszyn Boltzmanna

Praca magisterska na kierunku:
INFORMATYKA
Promotor:
**dr Tomasz Schreiber**

Toruń 2005

## Streszczenie

Praca przedstawia bibliotekę programistyczną umożliwiającą symulowanie prostych istot zwanych biotami. Bioty wyposażone są w zestaw elementarnych sensorów, efektorów, a także sieć neuronową kontrolującą ich zachowanie. Zadaniem biotów jest odkrycie strategii prowadzącej do chwilowego „zadowolenia", jakie daje wypełnienie prostego zadania, np. znalezienie pożywienia.

Poszczególne rozdziały opisują tło projektu. W pierwszym rozdziale przedstawiona jest dziedzina sztucznego życia, jej historia i projekty stanowiące inspirację dla biblioteki BrainLab. Drugi rozdział przedstawia metodykę uczenia ze wzmocnieniem, zwanego też uczeniem z krytykiem, jako technika uczenia maszynowego pokrewna wobec tego, co prezentują bioty. Trzeci rozdział traktuje o maszynach Boltzmanna, na których bazują bioty. Rozdział czwarty to opis samego projektu BrainLab, a pracę kończy rozważanie na temat możliwości rozwoju projektu.

## Słowa kluczowe

sztuczne życie, maszyny Boltzmanna, uczenie z krytykiem, a-life, bioty, biot

# Spis treści

# Wprowadzenie

Ta praca opisuje architekturę neuronową i bibliotekę programistyczną, stworzoną do symulacji autonomicznych biologicznie inspirowanych agentów - biotów w zmieniającym się środowisku.

Zarówno model neuronowy jak i oprogramowanie symulacyjne opierają się na pewnych restrykcyjnych regułach.

Po pierwsze - wszelkie decyzje były podejmowane na korzyść możliwości bardziej poprawnej biologicznie. Między innymi dlatego zdecydowałem się na maszyny Boltzmanna, które, choć nadal dalekie od neuronów biologicznych, mają pewne cechy obserwowane w układach nerwowych istot żywych.

Po drugie - wielką wagę przywiązuję do tego, by bioty były indywidualnymi bytami, uczącymi się na podstawie własnych doświadczeń, a nie obiektami sterowanymi przez jedną, centralną jednostkę.

Po trzecie - bioty powinny „wiedzieć” możliwie jak najmniej o ich położeniu na planszy, sposobie osiągnięcia celu, czy mierze odległości od celu lub stopnia wykonania zadania. To jest kluczowe założenie projektu, które definiuje jego innowacyjność.

Czwarta reguła jest po części powiązana z pozostałymi regułami - zarówno bioty z osobna, jak model środowiska zaprojektowany jest tak, by umożliwić w przyszłości implementację sprzętową. Celem symulacji jest umieszczenie architektury podejmującej decyzję w środowisku możliwie jak najbardziej podobnym do takiego, w jakim mógłby zostać uruchomiony mobilny robot wyposażony w proste zmysły.

ROZDZIAŁ 1

# Sztuczne życie

Dziedzina sztucznego życia (*artificial life*, *a-life*) to interdyscyplinarna dziedzina wiedzy, łącząca w sobie zagadnienia biologii, matematyki, psychologii, fizyki i wielu innych. Sztuczne życie polega na konstruowaniu maszyn i oprogramowania posiadającego pewne wybrane cechy istot żywych

Podczas gdy w dziedzinie AI - sztucznej inteligencji - zaczyna się od problemu (gra w szachy, rozpoznawanie języka naturalnego) i buduje system, który problem rozwiązuje, wykazując zaawansowane wzorce zachowania (podejście top-bottom), w dziedzinie sztucznego życia zaczyna się od symulacji i obserwuje się zachowania i problemy, jakie obiekt badań jest w stanie rozwiązać (podejście bottom-up). Taka organiczna metoda pozwoliła na konstrukcję (ewolucję) układów zdolnych do niezwykłych wyczynów w stosunku do prostoty budowy ich poszczególnych części. Można porównać to podejście do „odwrotnej inżynierii" (*reverse engineering*) - próby odgadnięcia zasad działania urządzenia, w tym przypadku umysłu istoty żywej, przez konstrukcję wzorowanego na nim modelu.

## 1.1. Samoreplikacja, automaty komórkowe i Życie Conwaya

Za pioniera dziedziny uważa się Johna von Neumanna, węgierskiego pochodzenia matematyka i pioniera teorii obliczeń. Von Neumann, student Turinga i Churcha, zadał pytanie: czy wystarczająco skomplikowana maszyna byłaby w stanie na podstawie własnego schematu skonstruować swoją kopię, lub nawet maszynę lepszą od siebie? W tym eksperymencie myślowym, von Neumann przekonał się, że taka maszyna, którą nazywał *Uniwersalnym Konstruktorem* (ang. Universal Constructor, inne nazwy to Clanking replicator i Auxon), nie łamie żadnych praw fizyki, a obecnie dysponujemy częścią technologii koniecznych do budowy Konstruktora. W eksperymencie myślowym von Neumanna, Uniwersalny Konstruktor byłby w stanie zbudować swoją kopię, która byłaby gotowa do budowy kolejnej kopii - i tak dalej. Taki samoreplikujący się zestaw Uniwersalnych Konstruktorów stale zwiększałby swoją moc produkcyjną, wykorzystując w tym celu odpowiednie surowce i energię, ale bez udziału człowieka - operatora.

Uniwersalny Konstruktor von Neumanna składał się z trzech elementów: Konstruktora $A$, Reproduktora $B$ i Kontrolera $C$. Konstruktor to maszyna, która jest w stanie zbudować dowolne urządzenie, o ile posiada instrukcję $F$ i odpowiednie zasoby. Reproduktor potrafi skopiować intrukcję, a Kontroler działa według schematu:

1. uruchamia Konstruktora, który wykonuje instrukcję $F$, zawierającą opis budowy zespołu $A + B + C$
2. uruchamia Reproduktora, by wykonał kopię instrukcji - $F'$
3. wyposaża nowo zbudowany zespół $A' + B' + C'$ w kopię instrukcji
4. oddziela zespół $A' + B' + C'$ wraz z instrukcją $F'$ od $A + B + C + F$.

(za [20])

Choć model w tej postaci nie zawiera możliwości samodoskonalenia (jak też nie spełnia innych zadań poza samoreplikacją), nic nie stoi na przeszkodzie, by uzupełnić kolonię Konstruktorów o algorytm genetyczny[1], który np. zapewnia przeżycie zespołów, które pracują szybciej i wykorzystują mniej zasobów.

Problematyka samoreplikujących się automatów była podejmowana przez ekspertów z NASA w raporcie z 1980 roku[11], redagowanym przez Roberta Freitasa. Raport dotyczył zautomatyzowanych misji kosmicznych, w których minimalny zestaw robotów wysłanych z Ziemi konstruowałby z zasobów naturalnych dostępnych na Księżycu czy Marsie kolejne roboty i elementy infrastruktury, którą w dalszej przyszłości mógłby zasiedlić człowiek. Projekt zautomatyzowanej sondy obejmował szczegóły działania takiego samobudującego się zespołu, od zasilania (zestaw baterii słonecznych), poprzez wytop metali, aż po metodę wytwarzania elementów elektronicznych. Badacze uznali jednak, że te ostatnie musiałyby być sprowadzane z Ziemi (na kształt witamin dla mechanicznego organizmu).

Najprawdopodobniej właśnie rozważania o samoreplikacji doprowadziły von Neumanna i Ulama do stworzenia teorii automatów komórkowych. Na automat komórkowy składają się: skończony zbiór $S$ stanów każdej pojedynczej komórki nieskończonej, skończeniewymiarowa (najczęściej jedno- lub dwuwymiarowa) siatka $S^d$, funkcja sąsiedztwa, która każdej komórce przypisuje zbiór komórek nazywanych jej sąsiadami oraz funkcja zmiany stanu, która wyznacza stan komórki w następnym kroku na podstawie stanu obecnego komórki i stanów komórek należących do jej sąsiedztwa. Aktualizacja stanów komórek może następować synchronicznie lub asynchronicznie. John von Neumann krótko przed śmiercią zdołał opisać 27-stanowy automat komórkowy zdolny do samoreplikacji.

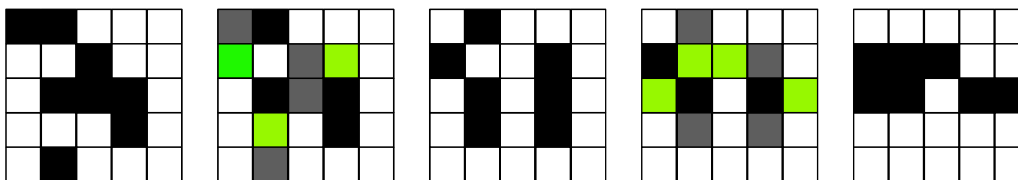
Automaty komórkowe to jeden z modeli obliczeń. Jako wygodny z matematycznego punktu widzenia model, używane są do symulacji rozprzestrzeniania się epidemii, badania właściwości fizycznych materiałów, przewidywania usterek w układach scalonych i w wielu innych dziedzinach, ale prawdziwa ich potęga objawia się nie w symulacji świata rzeczywistego, a w momencie, gdy potraktujemy je jako świat sam w sobie, w którym proste zasady rządzące zachowaniem jednostki prowadzą do niezwykle skomplikowanych zachowań.

[1] O algorytmach genetycznych - później

Najbardziej znanym przykładem automatu komórkowego jest gra Life (Życie), której stworzenie przypisuje się Johnowi Hortonowi Conwayowi zadebiutowała w roku 1970 w kolumnie gier matematycznych w Scientific American. Jest to gra bez graczy, w której stan początkowy jednoznacznie określa całą ewolucję układu. Układem jest nieskończona siatka dwuwymiarowa ($Z^2$), której każde pole jest automatem w jednym z dwóch stanów - żywy lub martwy. Gra oparta jest na dwóch prostych regułach:

- martwe pole (komórka), które ma dokładnie trzech żywych sąsiadów, ożywa
- żywa komórka pozostaje żywa pod warunkiem, że ma dwóch lub trzech żywych sąsiadów, w przeciwnym wypadku ginie (z samotności lub przeludnienia).

Sąsiedztwo w grze oznacza styczność brzegiem lub kątem: każda komórka ma ośmiu sąsiadów (sąsiedztwo Moore'a). Aktualizacje dokonywane są równolegle: stan komórki w kroku $n+1$ zależy od stanu sąsiedztwa w kroku $n$. Zakłada się, że konfiguracja początkowa jest skończona.

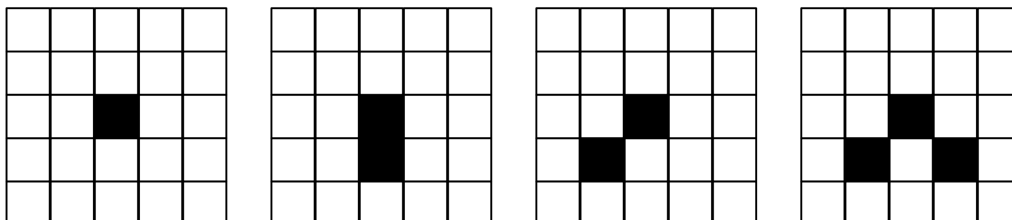


**Rysunek 1.1**. Konfiguracja początkowa i dwa kroki gry Life

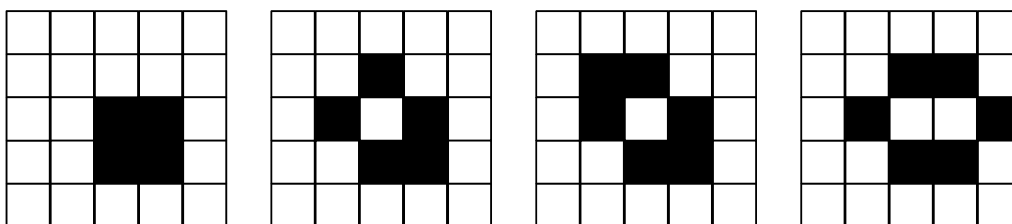
Martin Gardner, autor oryginalnego artykułu w Scientific American, przytoczył zalecaną przez Conwaya metodę grania za pomocą planszy i kamieni do gry Go, która zapewniała poprawność następnego kroku. Zaczynało się od na konfiguracji początkowej ułożonej z czarnych kamieni. Na pola, które miały umrzeć, należało położyć dodatkowy czarny kamień, a na te, na których miała się narodzić komórka - biały. Następnie należało upewnić się, że we wszystkich polach zostały uwzględnione reguły i przejść do następnego kroku, usuwając podwójne czarne kamienie i zastępując białe czarnymi. W czasach, gdy symulator Life uruchamia się na telefonie komórkowym, należy z szacunkiem wspomnieć zespół Conwaya żmudnie analizujący kilkudziesięciokomórkowe konfiguracje.

Gardner skategoryzował najprostsze konfiguracje początkowe, pokazując układy zamierające, zapętlające się i stabilne (lub stabilizujące się). Pojedyncza żywa komórka w oczywisty sposób zamiera w jednym kroku, podobnie dowolna konfiguracja dwóch komórek. Przykładem wzorca stabilnego jest kwadratowy blok czterech żywych komórek, a wzorca oscylującego - trzy żywe komórki ułożone w pionową lub poziomą linię (oscyluje między pionem a poziomem) lub dwa czterokomórkowe bloki stykające się rogami.

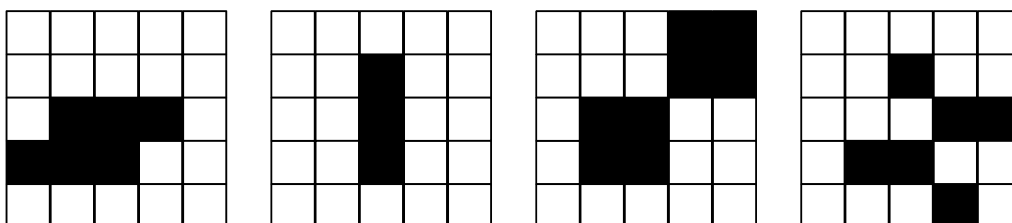
Układy oscylujące można podzielić na nieruchome, czyli takie, których kształt się zmienia, ale pozostają w miejscu i ruchome, zwane też statkami, czyli oscylatory, które wracają do wcześniejszego kształtu, ale z przesunięciem. Prędkość statków na planszy mierzy się w ułamkach $c$, czyli nieprzekraczalnej prędkości jednego pola na krok symulacji.



**Rysunek 1.2**. Proste wzorce zamierające.



**Rysunek 1.3**. Proste wzorce stabilne.



**Rysunek 1.4**. Proste oscylatory.

W grze Life, jak i w podobnych automatach komórkowych, *ogrodami Eden* nazywa się konfiguracje pierwotne, które nie mogą powstać z żadnej innej konfiguracji.

Oryginalny zamysł Conwaya był taki, aby żadna konfiguracja pierwotna w oczywisty sposób (w sensie prostoty dowodu) nie generowała populacji w nieskończoność, ale przy istnieniu nawet prostych konfiguracji początkowych, które rozwijają się przez dłuższy czas, zanim popadną w oscylację lub zaginą[2]. Rzeczywiście, eksperyment z komputerową wersją gry pokazuje, że losowo stworzona konfiguracja zwykle przez jakiś czas eksploduje

[2]Proste - do 10 komórek - konfiguracje początkowe, które generują bardzo duże populacje - od 100 komórek, nazywane były przez Conwaya Metuzalemami.

nowymi komórkami, po czym jej rozrost spowalnia i pozostają wysepki oscylujących konfiguracji. Conway ogłosił nagrodę 50 dolarów dla pierwszego, kto zaprojektuje konfigurację początkową, o której udowodni, że generuje ona nielimitowaną populację. Nagrodę odebrał Bill Gosper za projekt „strzelby” złożonej z dwóch stabilnych bloków i dwóch ruchomych oscylatorów, które „zderzając się” uwalniały bardzo prosty obiekt ruchomy zwany szybowcem. Powstało wiele podobnych konfiguracji, które podzielono na strzelby (oscylatory emitujące statki) i parostatki (statki emitujące spaliny).

Do dziś trwają konkursy na projekt oscylatora o możliwie największym okresie. Mając kilka oscylatorów (nie statków) o względnie pierwszych okresach, można je zsynchronizować tak, żeby stworzyć oscylator o okresie będącym iloczynem ich okresów[3]. Szczególnym osiągnięciem jest więc oscylator o okresie będącym liczbą pierwszą lub iloczynem niewielu liczb pierwszych. Uczestnicy takich konkursów lubują się w tworzeniu konfiguracji o artystycznych kształtach, przypominających przedmioty codziennego użytku.

Konkurs ogłoszony przez Conwaya pokazał, że w systemie, w którym identycznymi elementami rządzą proste reguły, z elementarnych części można projektować maszyny spełniające określone zadanie. W pełni deterministyczna gra bez zawodników, w której o wszystkim decyduje pierwotna konfiguracja, okazała się wdzięcznym tworzywem, w którym sama struktura obiektu decyduje o jego działaniu, a którego granice możliwości niełatwo było nawet oszacować. Przynajmniej do kwietnia roku 2000, kiedy Paul Rendell pokazał, że w modelu obliczeń, jakim jest automat komórkowy z regułami Life można zanurzyć maszynę Turinga. Konfiguracja Rendella jest złożeniem kilkudziesięciu części - bramek, elementów pamięci, implementacji stosu, taśmy i programu maszyny. Każda z nich zbudowana jest z prostych, sklasyfikowanych elementów - oscylatorów, elementów stabilnych i statków. Wszystkie elementy pozostają w interakcji. Realizacja programu zastępującego jedynkę na taśmie dwiema jedynkami, na danych wejściowych złożonych z dwóch jedynek kończy działanie po szesnastu krokach symulacji, zostawiając na taśmie cztery jedynki. Na komputerze używanym przez Paula Rendella trwa to ponad godzinę.

## 1.2. Modele ewolucyjne

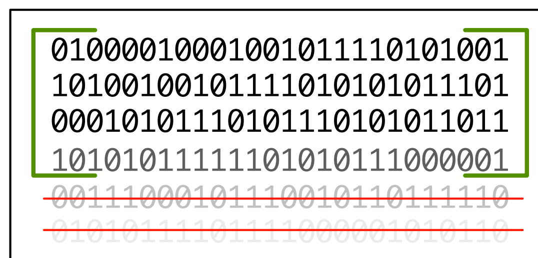
Automaty komórkowe, jak konfiguracje gry Life, choć podlegają rozmnażaniu i, jako całość, potrafią w trakcie rozwoju komplikować własną strukturę, nie dają możliwości arbitralnego określenia, co więcej - ustalenia miary, według której można mierzyć jakość kolejnych pokoleń. Konfiguracja gry Life równoważna maszynie Turinga jest prawdopodobnie ogrodem Eden, a nawet jeśli nie - nie sposób znaleźć konfigurację początkową, która mogłaby

[3]W konkursach bierze się pod uwagę konfiguracje spójne, czyli takie, w których nie można wyróżnić linii zawsze martwych komórek, po obu stronach której byłyby komórki żywe.

prowadzić do tak złożonej struktury. W grze Life i jej modyfikacjach, do stworzenia maszyny Turinga lub Konstruktora potrzebny jest inny Konstruktor - świadomy i inteligentny projektant. Filozoficzne implikacje tego spostrzeżenia byłyby ogromne, gdyby nie istniały *algorytmy genetyczne*, pozwalające ustalić kierunek mutacji (lub nawet nie) i iść na przysłowiową herbatę.

Na algorytm genetyczny składają się trzy elementy - *selekcja*, *krosowanie* i *mutacja* ([16], [10]. Podmiotem „obróbki ewolucyjnej" jest skończony zbiór genotypów, najczęściej reprezentowanych jako ciągi znaków. Dla ustalenia uwagi, pozwolę sobie ograniczyć się do przykładu, w którym genotypami są skończone ciągi zer i jedynek ustalonej długości (0100001000100101111010101001). Niech $G$ będzie zbiorem genotypów.

Selekcja to metoda oceniania genotypów. Biologicznym odpowiednikiem selekcji jest selekcja naturalna, czyli proces, który sprawia, że słabsze jednostki mają mniejsze szanse przekazać swoje geny. Najczęściej wymaga funkcji jakości lub przystosowania (*fitness*) $\phi : G \to \mathbb{R}$, która określa, które genotypy (a właściwie byty opisane przez te genotypy) są lepiej przystosowane (jeśli $g_1 \in G$ opisuje lepszą jednostkę niż $g_2 \in G$, to $\phi(g_1) > \phi(g_2)$). Mając funkcję przystosowania mamy kilka możliwości - możemy wybrać określoną ilość genotypów o najwyższych wartościach $\phi$, wybrać losowo (równoważnie - losowo odrzucić) pewną ilość genotypów z prawdopodobieństwem zależnym od $\phi$ (koło fortuny z nierównymi polami) lub zignorować funkcję przystosowania i odrzucić losowo wybrane jednostki.

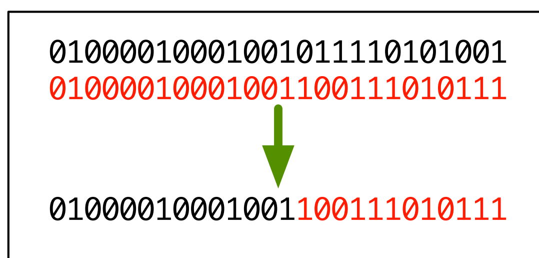



**Rysunek 1.5**. Selekcja

W świecie biologicznym funkcja przystosowania nie istnieje, choć można mówić o zmieniającym się w czasie rozkładzie prawdopodobieństwa przetrwania. To, czy dana jednostka doczeka się potomstwa, zależne jest od jej szansy przeżycia, stanu zdrowia i ogólnej sytuacji w otoczeniu: w epoce lodowcowej (lub podczas zimy, dla krócej żyjących stworzeń) większą szansę przeżycia mają jednostki lepiej utrzymujące ciepło i magazynujące energię. W czasie powodzi przetrwają te, którym nie przeszkadza woda, a w czasie suszy te, które szybko wodę znajdą lub stracą niewiele energii przed następnym deszczem - mikro-

ewolucyjne zmiany występują z pokolenia na pokolenie i są obserwowane[4]. Sam genotyp nie zawiera informacji, czy zostanie przekazany. W tym świetle na szczególną uwagę zasługuje metoda koła fortuny, jako najbardziej odpowiadająca temu, co wiemy o świecie biologicznym, a zarazem zapewniająca różnorodność puli genów.

Gdy wybraliśmy jednostki przeznaczone do przetrwania i odrzuciliśmy te, które nie miały szczęścia, trzeba wybrać jednostki (pary jednostek), które przekażą swoje geny. Tu różnorodność metod jest jeszcze większa - w niektórych modelach geny przekazują wszystkie jednostki, a kolejne pokolenie zastępuje poprzednie (lub jest do niego dodawane do kolejnej selekcji), w niektórych przekazanie genów zależy od położenia w symulowanej przestrzeni i bezpośredniego sąsiedztwa dwóch istot (jak w życiu), w niektórych przekazują je tylko jednostki najlepiej przystosowane. Metodę wyboru genotypów do krosowania warto dopasować do wybranego schematu selekcji - jeśli wcześniej zignorowaliśmy funkcję przystosowania, warto wykorzystać ją teraz, bo to ostatnia możliwość ukierunkowania procesu ewolucji i odróżnienia go od błądzenia losowego.




**Rysunek 1.6**. Krosowanie

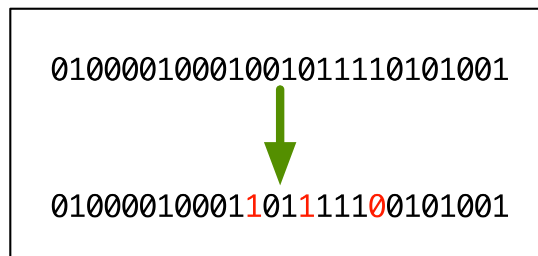
Krosowanie to proces w którym z dwóch (lub, bardzo rzadko, więcej) genotypów tworzony jest zbiór genotypów potomnych, najczęściej jeden lub dwa. W przypadku ciągów, których syntaktyka nie gra roli (każdy ciąg określonej długości nad danym alfabetem jest poprawnym genotypem), stosuje się krosowanie podobne do tego, które zachodzi przy łączeniu DNA dwóch osobników w genotyp nowego osobnika - łańcuchy są przecinane w ustalonym miejscu i w skład nowego łańcucha wchodzi pierwsza część jednego i druga część drugiego. Jeśli chcemy zachować poprawność składniową genotypów (w naturze najprawdopodobniej niepoprawne są po prostu odrzucane na pewnym etapie), trzeba zastosować inną metodę. W dość prostym przypadku, gdy genotypy można podzielić na atomowe podciągi - geny, z których każdy odpowiada jednej cesze, pilnuje się, aby miejsce dzielenia nastąpiło między podciągami, a nie pośrodku cechy. W przypadku drzewiastej interpretacji genotypu, podziału dokonuje się w samej interpretacji, według rozgałęzień.

[4]Mikroewolucja nie jest kwestionowana przez większość kreacjonistów, odrzucających ewolucję międzygatunkową

Gdy cechy kodowane sa liczbami rzeczywistymi, stosuje się np. średnią arytmetyczną do całości lub wybranych części genotypów.

Najważniejszym jednak wymogiem wobec krosowania jest to, aby potomek miał szansę zachować dobre cechy rodziców. Aby podać dość wyraźny przykład - jeśli o przystosowaniu w znacznym stopniu decyduje posiadanie w genotypie tej samej wartości na początku, co na końcu, a genotyp jest nad dość dużym alfabetem, krosowanie przez podział, z dwóch dobrze przystosowanych jednostek w większości przypadków wytworzy jednostkę zupełnie nieprzystosowaną. Przykład nie jest aż tak nieżyciowy jak by się mogło wydawać. Gdy mamy dany genotyp składający się z suboptymalnych rozwiązań problemu komiwojażera, kodowanych kolejnością odwiedzanych miast czy wierzchołków grafu, co powstanie ze skrzyżowania dwóch dróg, z których jedna jest odwróceniem kolejności drugiej? Za problem można tu winić zarówno nieoptymalne kodowanie, jak i destruktywną metodę krosowania. Projektując algorytm genetyczny należy dopasować metodę krosowania do kodowania cech w genotypie.

Mutacja odpowiada za spontaniczne generowanie cech nie występujących u przodków. W praktyce, jak w życiu, prawdopodobieństwo mutacji powinno być dość małe. najwyżej jeden gen, niewielka zmiana, w kilku genotypach na pokolenie. W niepełnych algorytmach genetycznych czasem pomija się krosowanie i opiera algorytm wyłącznie na selekcji i mutacji. W sytuacjach, gdy krosowanie z zachowaniem poprawności jest złożone obliczeniowo lub niemożliwe, jest to jedyny sposób.




**Rysunek 1.7**. Mutacja

Ogólny schemat postępowania w algorytmie genetycznym wygląda więc następująco:

1. zainicjalizuj populację
2. jeśli spełniony jest warunek zatrzymania, skończ
3. zastosuj na populacji selekcję i mutację
4. uzupełnij populację za pomocą operatora ktosowania

5. uaktualnij ocenę jakości osobników

6. wróć do pkt. 2

Algorytmy genetyczne, choć promowane przez zwolenników jako uniwersalna metoda, nie dla każdego problemu są najwydajniejszą metodą i zdecydowanie nie dla każdego problemu można sprecyzować rozwiązujący go algorytm genetyczny. Ich wielką zaletą jest jednak łatwość zrównoleglania. Zwykle najbardziej kosztownym obliczeniowo elementem jest funkcja przystosowania, a ta jest liczona osobno dla każdej jednostki. W dalszej części rozdziału nie będę opisywał rozlicznych zastosowań algorytmów genetycznych do rozwiązywania problemów optymalizacyjnych, a skupię się na kilku modelach sztucznego życia, obejmujących spontaniczną poprawę jakości jednostek przez algorytmy genetyczne w takiej lub innej postaci.

### 1.2.1. Życie, jakim mogłoby być

Thomas Ray, profesor biologii, zadał elementarne pytanie - czym właściwie jest życie[5]? Jedna z definicji używanych przez biologów opiera się na określeniu cech wspólnych tworów powszechnie uważanych za żywe, z zaznaczeniem, że istnieją wyjątki: istoty uznawane za żywe, które nie wykazują jednej lub kilku cech charakterystycznych dla istot żywych. Na naszej planecie nie potrzebujemy ścisłej definicji czy jasnego zestawu cech charakterystycznych, żeby określić czy próbka jest elementem biosfery czy atmosfery lub litosfery. Jednak w obliczu rozwoju egzobiologii, teoretycznej w każdym razie, pytanie Raya powraca i nabiera nowego znaczenia. Kolejne próbniki i sztuczne satelity wysyłane na najbliższą naszej planetę na której woda może występować w formie płynnej - Mars - zawsze zawierają czujniki wyspecjalizowane w wykrywaniu zjawisk sugerujących możliwość istnienia życia. Wyniki tych badań dotychczas jednoznacznie nie odrzuciły możliwości istnienia na Marsie prostych form, zdolnych do aktywnego podtrzymywania przetrwania tak jednostek jak i całego gatunku. Planując kolejne misje i projektując instrumenty, badacze stale zadają sobie pytanie - jakie cechy życia na Ziemi będą się przekładały na ewentualne życie na Marsie? A jeśli jednoznacznie udowodnimy istnienie na Marsie form, które będziemy zmuszeni uznać za zjawisko podobne do którejś z form życia ziemskiego, w jaki sposób zmieni się nasza definicja życia?

Ray, projektując pod koniec lat 80-tych symulację, którą nazwał Tierra, argumentował, że kolonie rozmnażających się agentów walczących o zasoby naturalne w wirtualnym środowisku nie są uproszczoną symulacją wykazującą zachowania podobne do życia biologicznego, a powinny być uznane za formę życia. Dla Raya symulowane środowisko ze swoimi zasobami naturalnymi nie jest modelem środowisk spotykanych na Ziemi, czy

[5] W znaczeniu - obszar badań biologii.

- być może - na Marsie, a formą świata, w którym może funkcjonować anatomicznie przystosowana do niego forma życia.

Zasoby naturalne w Tierra to czas procesora i pamięć - trzeba przyznać, że są to zasoby bardzo *naturalne* dla symulacji komputerowej, a także skończone i w pewnym sensie cenne. Istoty korzystające z takich zasobów naturalnych to ciągi instrukcji w prostym języku programowania, obejmującym interakcję między programami, między programem a otoczeniem oraz zdolność samokopiowania. Algorytm genetyczny nie był oparty na zaprojektowanej funkcji przystosowania - przetrwanie i przekazanie genów zależało od działania samego programu, mutacja polegała na dodawaniu i usuwaniu fragmentów programu, co zachodziło z rozkładem Poissona. Programy były wykonywane w systemie czasu dzielonego - program otrzymywał dość krótkie okno czasowe, możliwość realizacji kilku swoich poleceń i sterowanie przekazywane było do innego. Program niezdolny zarezerwować sobie odpowiedniej ilości pamięci do przetrwania czy umieszczenia w niej funkcjonalnej kopii samego siebie. Pomysł Raya, oparty na walczących ze sobą programach komputerowych nie jest ani pierwszym, ani ostatnim projektem tego typu - należy wspomnieć tu projekt CoreWars, czy bazujący na Tierra świat Avida autorstwa Chrisa Adami.

System Tierra, zapłodniony jedną zaprojektowaną istotą (jedyną istotą zaprojektowaną przez człowieka), z czasem staje się ekosystemem pełnym zróżnicowanych stworzeń, które rozmnażają się, mutują i pozostają w złożonej interakcji. Ray zauważył istoty o pasożytniczym trybie życia: wykorzystujące fragmenty kodu innych istot, by się rozmnażać.

Idea samodoskonalącego się kodu programu komputerowego może się okazać zbawieniem wobec faktu, że obecnie największy koszt przy produkcji oprogramowania ponosi się na etapie testowania. Program testuje się przez uruchomienie go w możliwie największej liczbie różnych sytuacji: dla różnych danych wejściowych, scenariuszy użytkowania, przy różnych konfiguracjach sprzętowych. Samodoskonalący się program lub komitet ewoluujących programów mógłby, teoretycznie, dostosowywać się do nowych sytuacji tak, by jak najlepiej spełniać wymogi specyfikacji. Taki program od programisty wymagałby tylko (albo aż) napisania specyfikacji. W ogólnym przypadku można się spodziewać, że stosowana obecnie metoda jest bardziej optymalna - nikt nie chce, aby jego ulubiony edytor tekstu nagle zmutował, rezygnując z użytkownika jak z niepotrzebnej kończyny. Nie należy jednak odrzucać możliwości „hodowania" elementów programu.

### 1.2.2. Framsticks

Framsticks to opracowane przez Macieja Kosmosińskiego i Szymona Ulatowskiego oprogramowanie modelujące ewolucję trójwymiarowych stworzeń (zwanych niekiedy framstickami). Projekt, od swojego oficjalnego powstania w roku 1997 był wielokrotnie nagradza-

ny i doczekał się wielu publikacji. Kolejne moduły i programy towarzyszące są rozwijane przez zespoły programistów mniej lub bardziej związanych z oryginalnymi twórcami.

Framsticks modeluje trójwymiarowy teren, na którym umieszczane są istoty o budowie i umiejętnościach ściśle ustalonych przez ich indywidualne genotypy. Zależnie od modelowanego terenu i ustalonej miary jakości (*fitness*), istoty ewoluują w biegające jaszczurki, majestatycznie pływające meduzy lub wysokie drzewa - wszystko zależy od parametrów symulacji i poświęceonego czasu procesora. Oczywiście, wprawiony w nietypowych językach programowania badacz dziedziny może zaprojektować i poddać symulacji własnego framsticka.

Genotyp framsticka może być spisany w jednym ze stworzonych do tego celu języków programowania (raczej: formatów genotypów), oznaczanych `f0`, `f1`, `f2` i tak dalej. Różnią się one restrykcyjnością w projektowaniu, jak też zakresem, np. `f4` obejmuje nie tylko budowę, ale też proces wzrostu pojedynczego organizmu. Formaty `f5` oraz `f6` opisują chemię i metabolizm framsticków. Format `f7` to format przyjmujący dowolny ciąg znaków - łatwo poddaje się mutacji. W praktyce stosowane są trzy formaty - `f0`, jako najbardziej elementarny, głównie wewnętrznie w oprogramowaniu i do tworzenia bardziej fantazyjnych framsticków, `f1`, jako najprostszy, pozwala szybko tworzyć duże framsticki, oraz `f4`, do realistycznych, zaplanowanych symulacji, w których oczekujemy, żeby framsticki rosły. Dostępny jest jeszcze format `f0Fuzzy`, pozwalający na rozwijanie wewnątrz framsticków kontrolerów na logice rozmytej.

Format zero, `f0`, określa szczegółowo budowę framsticka w danym momencie rozwoju (o ile ma on możliwość rozwoju). Jest to format najmniej restrykcyjny i pozwala na budowanie szczegółowych, trójwymiarowych grafów, tworzących piramidy, wieże czy realistycznie wyglądające (i zachowujące się) organizmy. Jednego framsticka w formacie `f0` określa program składający się z ciągu linijek postaci:

```
KLASA:WŁASNOŚĆ1,WŁASNOŚĆ2, ...
```

gdzie `KLASA` to ustalony identyfikator klasy (np. staw oznaczany jest literą `j`), a `WŁASNOŚĆ` to specyfikacja własności danej klasy (położenie w przestrzeni, masa, rozmiar) i może mieć postać `NAZWA=WARTOŚĆ`, `WARTOŚĆ` lub postać pustego ciągu, oddzielonego przecinkami od sąsiadów. Nazwę własności można pominąć przy określaniu pierwszej własności, oraz własności następnej w naturalnej kolejności po dopiero określonej. Wartość może być liczbą naturalną, zmiennoprzecinkową lub ciągiem znaków - w takim wypadku objęta musi być cudzysłowami. Jeśli dana własność nie jest wymieniona w specyfikacji, otrzymuje wartość domyślną. Tak więc, poprawne są linijki postaci:

```
klasa:a=1,b=2,c=3,d=4
klasa:1,2,3,4
```

```
klasa:d=4,a=1,2,3
klasa:1,,4
klasa:c=",",d=3
klasa:
```

Niezależnie od wartości domyślnych dla własności `a`, `b`, `c` i `d` w klasie `klasa`, pierwsze trzy z wymienionych linijek są równoważne. Własności określające położenie obiektu w przestrzeni określane są najczęściej przez `x=10,30,20` zamiast `x=10,y=30,z=20`.

Klasy w formacie zero obejmują części konstrukcyjne framsticków (podłużne belki, oznaczane `p` od *part*), stawy (jak wspomniałem, oznaczane `j`, od *joint*), neurony (`n`) oraz wejścia neuronów (`c`, *connection*). Klasa `p` ma własności określające jej położenie w przestrzeni (`x`, `y` oraz `z`), obrót w trzech osiach (`rx`,`ry`,`rz`), masę (`m`), rozmiar (`s`), gęstość (`dn`), współczynnik tarcia (`fr`), zdolność wytwarzania energii z pożywienia (`ing`), oraz fotosyntezy (`as`). Staw, oprócz wskaźników na łączone części konstrukcyjne i współrzędnych obrotu, dysponuje parametrem sztywności przy zginaniu (`stif`) i obrocie (`rotstif`) i wytrzymałością przy zderzeniach (`stam`). Samo wyliczenie cech tych komponentów daje dobre pojęcie o możliwościach środowiska symulacyjnego.

Każdy framstick może (nie musi) dysponować siecią neuronową. Neuron, oprócz własności `p` i `j` określających część i/lub staw, z którym jest połączony, dysponuje własnością `d`, której wartością może być ciąg znaków określających neuron. Opis w wartości własności `d` może dotyczyć neuronu przetwarzającego sygnał, sensora lub mięśnia. Neurony przetwarzające sygnał dzielą się na sigmoidalne (`N`), róznicujące (`D`), progowe (`Thr`), stałe (`*`), losowe (`Rnd`), sinusoidalne (`Sin`) oraz rozmyte (`Fuzzy`). Każdy z nich dysponuje zestawem odpowiednich sobie parametrów. Mięśnie i sensory to: zginacz (`|`), obracacz (`@`), czujnik żyroskopowy (`G`, jego wartością jest tangens kąta nachylenia odpowiadającej mu części konstrukcyjnej), czujnik dotykowy (`T`), węch (`S`, reaguje na twory obdarzone energią), oraz czujniki wody (`Water`) i energii wewnętrznej framsticka (`Energy`). Przykładowo, ciąg `Sin,f0:0.1,t:0.5` opisuje neuron generujący na wyjściu falę sinusoidalną o częstotliwości 0.1 i fazie 0.5.

Obiektowi każdej wymienionej dotychczas klasy można jeszcze przypisać indywidualny opis, w postaci ciągu tekstowego pod własnością `i`. Własności opisu nie posiada klasa „synaptyczna" (`c`), oprócz pary neuronów, których dotyczy (neuronu `n` oraz numeru jednego z jego wejść `i`), pozwala ustalić wagę połączenia: `w`.

Genotyp musi spełniać kilka warunków poprawności, np. każdy staw musi mieć przypisane dokładnie dwie części, wszystkie części muszą tworzyć graf spójny, a przesunięcia i obroty na stawach nie mogą powodować konfliktów, jednak format `f0` daje badaczowi ogromne możliwości. W oprogramowaniu podstawowym językiem dostępnym dla operatora jest `f1`, więc genotypy w `f0` oznacza się ciągiem `//0` w pierwszej linijce.

W formacie zero poprawne są następujące genotypy:

Trzy części połączone w ciąg:

```
p:
p:1,m=2
p:2,m=2
p:3,
j:0,1
j:1,2
j:2,3
```

Prosty przykład z siecią neuronową:

```
p:
p:1
j:0, 1, dx=1
n:p=1
n:j=0, d="|:p=0.25,r=1"
n:j=0, d=G
n:p=1
n:j=0, d=@:p=0.25
n:p=1, d=T
c:0, 2
c:0, 3, 2.3
c:1, 0
c:3, 0, 3.4
c:3, 3, 4.5
c:3, 5, 5.6
c:4, 3
```

Cztery części tworzące kwadrat - struktura niemożliwa do uzyskania w formatach `f1` czy `f4`:

```
p:0,0
p:1,0
p:1,1
p:0,1
j:0,1
j:1,2
j:2,3
j:3,0
```

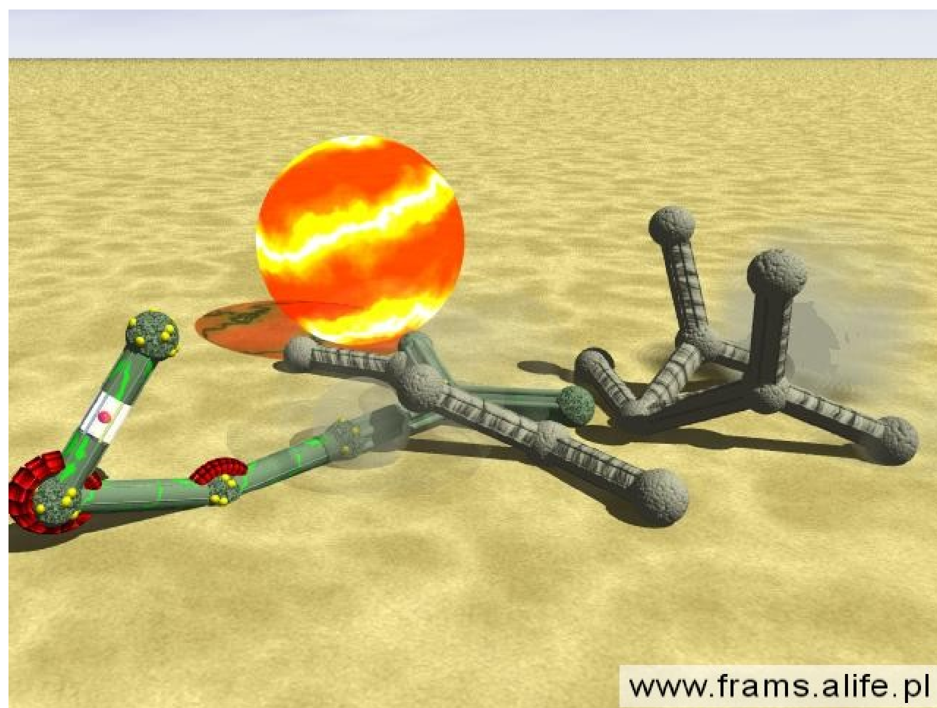
Format `f1` to język do szybkiego opisu framsticków. Programista nie musi koncentrować się na częściach i stawach z osobna, ponieważ do każdej części automatycznie dodawany jest staw. Połączenia między neuronami są ustalone na zasadach sąsiedztwa, a wartości ustalane liczbowo w `f0` ustala się za pomocą jednoznakowych modyfikatorów. Drzewiastość możliwych do skonstruowania struktur sprawia, że cały genotyp składa się z jednej (zwykle dość długiej) linijki.

Składnia `f1` obraca się wokół gałęzi (`X`) oraz rozgałęzień, oznaczanych nawiasami okrągłymi. O ile nie zostanie to określone inaczej, gałęzie wchodzące w skład rozgałęzienia, dzielą się po równo przestrzenią kątową na wspólnej płaszczyźnie. Trzy gałęzie (`X(X,X)` utworzą coś w rodzaju litery *Y* (choć do *Y* bardziej podobny jest genotyp `XX(X,X)`). W praktyce nie jest istotne która z gałęzi jest „rodzicem" rozgałęzienia, genotyp opisuje po prostu graf bez cykli. W genotypie opisującym strukturę podobną do pieska pierwszy symbol `X` może oznaczać prawą tylnią łapę lub końcówkę ogona.

Dwa przecinki pod rząd wewnątrz rozgałęzienia sprawiają, że algorytm rozdzielania przestrzeni kątowej zachowuje się, jakby była między nimi kolejna gałąź, ale jej nie umieszcza: genotyp `X(X,,X)` opisuje strukturę przypominającą literę *T* (znowu: literę *T* przypomina bardziej inny genotyp, `XX(X,,X)`). Ogólnie - dwa `X` obok siebie zawsze oznaczają połączone ze sobą części konstrukcyjne, a dwa `X` oddzielone przecinkami i otoczone nawiasami oznaczają rozgałęzienie wychodzące z tego, co znajduje się po lewej stronie nawiasu.

Własności poszczególnych gałęzi (czyli składających się na nie części konstrukcyjnych i stawów) określają modyfikatory, występujące przed symbolami `X`. Modyfikatory to pojedyncze symbole, wielkie lub małe litery. Wielka litera oznacza zwiększenie własności w stosunku do jej wartości domyślnej, mała - zmniejszenie, np. wielka litera `R` obraca gałąź o 45 stopni w jedną stronę, mała - w lewą. `Q` - wykręca gałąź, wraz z gałęziami połączonymi z nią na jednym z końców (obrót za pomocą `R` nie ma tej własności). `C` zgina staw na jednym z końców (paradoksalnie, genotyp `CXX` oznacza: gałąź, zgięcie, gałąź). Modyfikatory `L`, `W` oraz `F` kontrolują własności fizyczne gałęzi - długość, wagę i tarcie, odpowiednio. Istnieją również modyfikatory określające funkcję biologiczną gałęzi, wzajemnie wykluczające się - gałąź może albo fotosyntetyzować (`A`), albo pełnić funkcję obronną (`S`), mieć wzmożoną siłę i szybkość mięśnia (`M`) lub wchłaniać pożywienie (`I`). Zmusza to projektanta do oszczędnego gospodarowania zasobami strukturalnymi framsticka. Przykładami poprawnych genotypów są: `XXX(XX,X)`, `X(X,RRX(X,X))`, `XlCXlCXlCX`.

Sieć neuronową framsticka w formacie pierwszym buduje się przez umieszczenie opisów neuronów w nawiasach kwadratowych *po* symbolach odpowiadających im gałęzi: genotyp `LLLLX[N]` oznacza przedłużoną gałąź z przypisanym jej pojedynczym neuronem sigmoidalnym. Pojedynczej gałęzi może być przypisane kilka neuronów, w sąsiadujących nawiasach kwadratowych. Każdy genotyp w `f1` można przetłumaczyć na `f0`.




**Rysunek 1.8**. Trzy framsticki

Format czwarty jest pochodną formatu drugiego, opisującego wyłącznie łączenie części, i trzeciego, opisującego wyłącznie zasady rozbudowy framsticka. Przypomina f1, ale zamiast określać anatomię framsticka, zawiera instrukcje do poszczególnych jego części - jak mają się rozrastać. Części wykonują te instrukcje równolegle, a algorytm kończy się, gdy wszystkie części zakończą swoje „programy”. Częściami konstrukcyjnymi w f4 są komórki, które mogą być gałęziami, neuronami lub komórkami macierzystymi, zdolnymi przekształcić się w neuron lub gałąź. Symulacja rozpoczyna się od pojedynczej komórki macierzystej. Najprostszym przykładem genotypu w f4 jest /*4*/X, generujący w jednym kroku framsticka składającego się z pojedynczej gałęzi.

Trudniejszy i zarazem bardziej interesujący przykład: /*4*/<X>X. Symbole < oraz > nie są nawiasami kątowymi otaczającymi pierwszy symbol X, to instrukcje dla komórek macierzystych. Proces rozwoju wygląda tu następująco:

1. początkowa, jedyna komórka macierzysta wykonuje instrukcję < (podziel się) i tworzy nową komórkę macierzystą

2. ta sama, pierwsza komórka wykonuje instrukcję X i zamienia się w gałąź, równolegle - nowa komórka macierzysta wykonuje drugą instrukcję X i zamienia się w gałąź

3. pierwsza komórka natrafia na instrukcję > (skończ), druga nie znajduje dla siebie instrukcji, więc również kończy rozwój.

Oznaczając pierwszą komórkę macierzystą jako ., adrugą przez ,, można ten proces przedstawić następująco:

1. .<X>X - komórka macierzysta znajduje swoją instrukcję - <
2. .X>,X - nowa komórka macierzysta znajduje koniec instrukcji swojej komórki rodzicielskiej
3. X>X - obie komórki przyjęły postać gałęzi (X)
4. XX - obie komórki skończyły rozwój.

Formalnie, każdy genotyp w f4 pownien kończyć się symbolem końca rozwoju, jednak symbol ten można pominąć, dzięki czemu genotyp wygląda bardziej estetycznie - z tą samą ilością <, co >. Instrukcje dla komórek to:

- X - zamień się w gałąź (po zamianie w gałąź komórka nie może wykonywać instrukcji podziału (<)
- N - zamień się w neuron (sensor, efektor)
- , - zwiększ kąt rozgałęzienia, równoważne dopisaniu , wewnątrz nawiasów w f1
- kody odpowiadające modyfikatorom gałęzi - komórka je po prostu przepisuje
- kody mięśni - neuron staje się neuronem odpowiedniego typu
- # - znacznik powtarzania, po nim następuje ilość iteracji i ciąg instrukcji zakończony >

Niezależnie od wybranego języka w którym opisywane są genotypy, symulowane istoty umieszcza się w zaprojektowanym lub losowo wygenerowanym kwadratowym środowisku, obejmującym nierówności terenu, wodę i kulki pożywienia. Format pliku, w którym dostarcza się danych do symulacji obejmuje znacznie więcej niż graficzny interfejs użytkownika i umożliwia ręczne tworzenie bardzo skomplikowanych symulacji i skryptów modyfikujących warunki w czasie rzeczywistym.

Na podstawową symulację Framsticks składa się pula genów - zbiór stworzeń napisanych w jednym z formatów f0, f1 lub f4 i zasady algorytmu genetycznego. Pula genów musi składać się z przynajmniej jednej jednostki - wytworzonej „z ręki" lub powstałej wskutek mutacji z prostszego stworzenia (co nie znaczy, że mutacja zawsze zwiększa komplikację genotypów).

Przed rozpoczęciem symulacji określa się maksymalną wielkość puli genów, metodę selekcji i wzór na funkcję przystosowania. Metoda selekcji może być jedną z trzech - losowe

odrzucanie jednostek, odrzucanie najsłabszych lub odrzucanie według prawdopodobieństwa - nierówne koło fortuny. Nowe genotypy powstają wskutek mutacji (dopisywanie lub usuwanie fragmentów genotypów z zachowaniem poprawności syntaktycznej) i krzyżowania genotypów pozostających w populacji. O krzyżowaniu decyduje nie tylko funkcja przystosowania, ale też stopień podobieństwa rodziców - autorzy w ten sposób unikają problemu tracenia dobrych cech, o którym wspominałem.

Funkcję przystosowania określa się na podstawie średniej ważonej parametrów framsticka - prędkość, wysokość, czas życia (ma znaczenie jeśli na planszy istnieje pożywienie, a framstick potrafi je znajdować i spożywać). Odpowiednio nią manipulując, można wytworzyć istoty pnące się ku górze, szybko poruszające się, wytrzymałe na uderzenia, lub przejawiające po części wszystkie te cechy. Ewolucja framsticków trwa jednak niezwykle długo, co jest spowodowane metodą obliczania funkcji przystosowania - jest ona szacowana gdy istota wyczerpie swoją energię, co może trwać dość długo u dobrze przystosowanych framsticków. Dostępny jest projekt rozproszonego ewoluowania framsticków.

## 1.3. Modelowanie zachowania

Modele istot w Framsticks mają proste sieci neuronowe, pozwalające im reagować na zmiany w otoczeniu i wykonywać cyklicznie powtarzające się ruchy. Teoretycznie możliwe jest, aby wyewoluowała kreatura zdolna przetrwać w nieskończoność w niesprzyjającym terenie, o ile pożywienia będzie regularnie przybywało - jej głównym zajęciem byłoby unikanie niebezpieczeństwa i znajdowanie pożywienia framstickowym zmysłem węchu energetycznego, jednak taka symulacja niesie ze sobą pewne ograniczenia. Złożona fizyka świata zajmuje większość czasu procesora, a ukierunkowanie oprogramowania na algorytmy genetyczne czyni modele uboższymi - framsticki są ograniczone do tego, co można zapisać w ich genotypach.

Tworzenie sztucznego życia ma na celu poznanie zasad funkcjonowania organizmów żywych nie tylko w kontekście rozmnażania się, ale też zachowań. Skąd pojedyncza ryba w ławicy wie, że stanowi element wielkiej, błyszczącej formacji, która ma na celu zmylenie drapieżnika? Gdzie jest termit-architekt, który wie jak ma wyglądać kopiec?

Subtelna różnica między metodologią sztucznego życia, a sztucznej inteligencji polega na podejściu do problemu. W sztucznej inteligencji szuka się rozwiązania problemu przez analizę celu, czy celem jest analiza języka naturalnego, czy gra w szachy. W a-life stosuje się podejście eksperymentalne. Wiedząc, że problem posiada rozwiązanie w naturze i obserwując sposób, w jaki problem jest rozwiązany, konstruuje się symulację i bada się, czy symulowane „istoty” rozwiązują problem, a także - czy przy uproszczeniach ko-

niecznych do przeprowadzenia symulacji, pojawiają się pewne dodatkowe, skomplikowane zachowania, tzw. zachowania wynikające (*emergent behaviour*).

W 1986 roku, Craig Reynolds zaprezentował wyniki eksperymentu, który odbił się echem zarówno w dziedzinie sztucznego życia, co w dziedzinie animacji komputerowej. W trójwymiarowej przestrzeni symulacji umieścił kolonię identycznych agentów, zwanych „boids”[6], dając każdemu możliwość poruszania się w przestrzeni i identyczny zestaw instrukcji:

- unikaj zderzeń
- podążaj w tym samym kierunku co sąsiedzi
- trzymaj się w pobliżu sąsiadów

Sąsiedztwem boida są wszystkie boidy znajdujące się bliżej niż pewien ustalony promień sąsiedztwa, z wyłączeniem tych znajdujących się za nim, wewnątrz pewnego kąta - obie wartości są modyfikowalnymi parametrami w symulacji. Boid nie może się zatrzymać, może jedynie modyfikować kierunek swojego lotu. Modyfikacja następuje w każdym kroku symulacji i jest wypadkową wektorów wynikających z trzech reguł sterowania.

Te trzy proste reguły zaowocowały niezwykle realistyczną symulacją przypominającą stado ptaków lub ławicę ryb. Zachowaniem wynikającym jest tu tworzenie spójnego stada - z reguł rządzących zachowaniem jednostki wynika zachowanie grupy. Traf chciał, że kilka miesięcy po publikacji pracy Reynoldsa, Chris Langton organizował warsztat - Artificial Life Workshop. Projekt boids jest od tej pory uznawany za wzorcowy projekt z dziedziny sztucznego życia.

Reynolds zastosował technologię w krótkim filmie animowanym pt. „Stanley and Stella”. Później metoda modelowania zachowania została zastosowana do renderowania stad nietoperzy i człapiących przez miasto pingwinów w filmie „Powrót Batmana”, galopujących antylop w „Król Lew” i tłumów ludzi w „Mulan”.

Projekt boids doczekał się wielu modyfikacji - sam Reynolds uzupełnił model o regułę dążenia do celu i unikania nieruchomych przeszkód. Stado, wobec niebezpieczeństwa zderzenia, rozdzielało się na dwie części, które w płynny sposób łączyły się w jedną większą grupę. Inne modele uwzględniały podążanie za przodownikiem stada, ucieczkę przed drapieżnikiem, śledzenie wyznaczonej linii, grę w berka, ustawianie się w kolejce, ucieczkę z pomieszczenia z jednym wyjściem i wiele innych. W każdym modelu złożona koordynacja stadnego zachowania następuje bez odgórnej kontroli, czy nawet komunikacji między jednostkami.

Oprogramowanie *Breve* zapoczątkowane przez Jona Kleina jako element pracy magisterskiej umożliwia konstruowanie symulacji zachowań stadnych przy założeniu dowol-

[6]Dziecięca wymowa angielskiego słowa *bird* - ptak.

nych reguł, z agentami, które mogą być pojedynczym punktem, a mogą być złożonym z kilku brył tworem na kształt framsticka. Symulacja może generować dźwięk, obraz, wykorzystując agentów jako kolonię wirtualnych pędzli, a także implementować algorytm genetyczny. Ciekawą symulacją dodaną do programu jako przykład jest symulacja stada zbieraczy. Zbieracze poruszają się w trójwymiarowym środowisku, przenosząc obiekty, które można utożsamiać z pożywieniem. Rządzą nimi cztery reguły:

- jeśli nie masz w otoczeniu pożywienia - błądź losowo
- jeśli widzisz pożywienie, a jeszcze nie przenosisz - złap
- jeśli przenosisz pożywienie, a nie widzisz w otoczeniu - błądź
- jeśli przenosisz pożywienie, a znajdziesz się w miejscu, gdzie jest go dużo - zostaw.

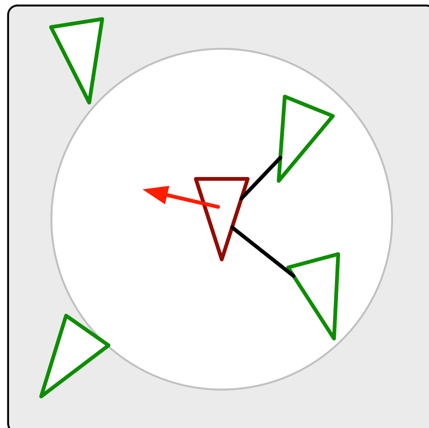
Błądzenie losowe w tej symulacji oznacza orbitowanie wokół środka ciężkości całego zbioru pożywienia z drobnymi, losowymi zmianami kierunku ruchu, dzięki którym zbieracz nie porusza się stale w tej samej płaszczyźnie. Po dłuższym czasie symulacji można zaobserwować, że pożywienie, początkowo rozłożone równomiernie po przestrzeni symulacyjnej, zaczyna formować wysepki. Stopniowo, mniejsze wysepki zaczynają znikać, a większe powiększają się jeszcze bardziej, aż pożywienie tworzy jedno zagęszczenie, a zbieracze przenoszą je w tę i z powrotem. Zachowaniem wynikającym jest tu pozornie skoordynowane tworzenie zbiorowiska, bez wcześniej przygotowanego planu.

## 1.4. Wnioski

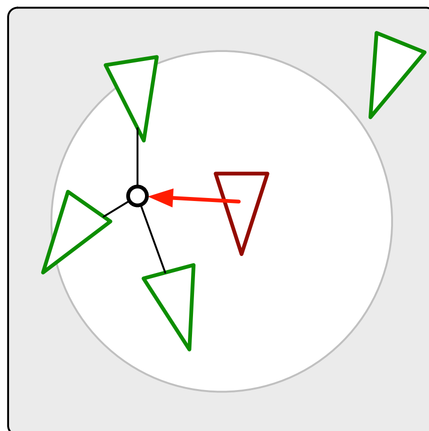
Dziedzina sztucznego życia jest rozwijającą się gałęzią wiedzy, co widać po różnorodności modeli. Należy tu zaznaczyć, że wymienione wcześniej projekty nie stanowią przeglądu dziedziny, a tylko linię genealogiczną od modeli obliczeń opartych na rozważaniach symulowania życia, poprzez ukucie terminu *sztuczne życie*, po modele, które stały się bezpośrednią inspiracją dla projektu BrainLab. W dziedzinie sztucznego życia dzieje się znacznie więcej - od fotorealistycznej symulacji wzrostu roślin, po symulacje całych ekosystemów, pozwalające na przewidywanie wpływu czynników środowiskowych na liczebność gatunków na danym terenie.

Symulacje z dziedziny sztucznego życia z biegiem czasu stają się coraz bardziej złożone, coraz bardziej zaczynają przypominać biologiczne stworzenia, nie tylko spełniając charakterystyczne dla nich funkcje, ale też wizualnie - gwałtowny przyrost dostępnej mocy obliczeniowej pozwolił na symulację istot o choreografii godnej roli w filmie. Modele zbudowane, by funkcjonować w fizycznym świecie, jak małe roboty tworzone przez Rodneya Brooksa, znajdują zastosowanie w zadaniach wymagających wykonania prostej czynności

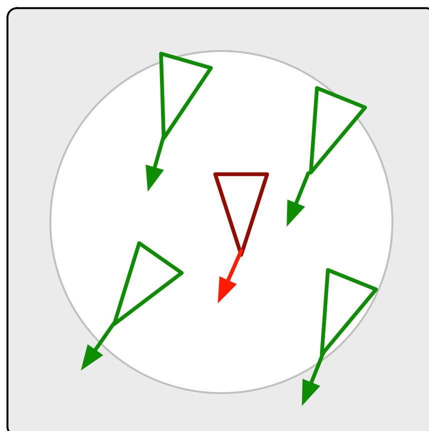
na dużym terenie w możliwie krótkim czasie - jak sprzątanie czy znajdowanie min. Od czasu teoretycznych rozważań Newmanna i gry Conwaya, biologicznie inspirowane modele interakcji i rozwoju jednostek pośrednio lub bezpośrednio prowadziły do rozwoju nowych technologii i dziedzin wiedzy, co jest tylko naturalną kontynuacją tradycji podpatrywania natury przy projektowaniu nowych wynalazków.



**Rysunek 1.9**. Boid ma unikać zderzeń.



**Rysunek 1.10**. Boid ma trzymać się blisko stada.



**Rysunek 1.11**. Boid ma kierować się w tym samym kierunku co sąsiedzi.

ROZDZIAŁ 2

# Uczenie ze wzmocnieniem

## 2.1. Uczenie maszynowe

Uczenie maszynowe stosujemy tam, gdzie nie jesteśmy w stanie znaleźć analitycznego rozwiązania problemu lub tam, gdzie dokładna, algorytmiczna metoda jest bardziej kosztowna obliczeniowo niż heurystyka oparta na uczeniu. Im częściej oczekujemy od maszyn i oprogramowania aktywnej interakcji z użytkownikiem lub otoczeniem, tym częściej natrafiamy na tego typu problemy - zestawy danych statystycznych nie generowane, a zbierane w świecie rzeczywistym, np dane medyczne, mają bardzo nietypowe rozkłady, co uniemożliwia jednoznaczne określenie reguł pozwalających klasyfikować pojedyncze przypadki. Niekiedy głównym utrudnieniem jest zmienność warunków w czasie - problem klasyfikacji spamu i innej niechcianej poczty nie daje się rozwiązać pojedynczym algorytmem, ponieważ twórcy masowej poczty szybko dopasowują jej treść tak, by przypominała normalne wiadomości. Alternatywą dla stałego projektowania nowych algorytmów jest algorytm uczący się, który z pewną pomocą użytkownika stale dopasowuje się do nowych wynalazków spammerów.

Co to znaczy, że dany system się uczy? Tom Mitchell w [9] pisze, że uczenie maszynowe to dziedzina wiedzy dotycząca algorytmów, które automatycznie poprawiają się przez doświadczenie. Z ogółu modeli uczących się można wywnioskować, że uczenie maszynowe polega na ukierunkowanej modyfikacji parametrów systemu na podstawie danych, przy ustalonej mierze jakości uczenia. Mamy więc dane, parametry systemu i metodę oceny, czy uczenie się udało.

Zależnie od sformułowania problemu i postaci oczekiwanej odpowiedzi, wyróżnia się kilka modeli uczenia maszynowego.

Uczenie *bez nadzoru* dotyczy między innymi sieci samoorganizujących się. Zadaniem sieci jest dopasować swoją strukturę do danych tak, by odzwierciedlić ich strukturę - nie tylko podzielić na pewną ilość kategorii, ale też wykazać zależności między kategoriami.

Uczenie *z nauczycielem* polega na podsuwaniu systemowi uczącemu się danych wejściowych i oczekiwanego wyjścia tak długo, aż dokładność odtwarzania będzie zadowalająca. Oczekiwanym od tak uczonych systemów zachowaniem jest generalizacja, czyli zadowalające odtwarzanie poprawnych odpowiedzi dla wzorców, które nie znajdowały się w zbiorze treningowym.

Co jednak jeśli nie znamy poprawnej odpowiedzi (albo znamy, ale chcemy, aby uczący

się agent sam ją odkrył), ale jesteśmy w stanie określić jaka odpowiedź czy zachowanie jest odpowiednie lub zadowalające?

## 2.2. Procesy decyzyjne Markowa

*Procesem decyzyjnym* nazywamy wędrówkę agenta[1] podejmującego decyzje o poruszaniu się w zmiennym środowisku i otrzymującego za każdym krokiem pewne liczbowe wynagrodzenie. Dla agenta szukającego wyjścia z labiryntu, wynagrodzeniem jest odległość od wyjścia. Dla eksperymentującego komiwojażera, wynagrodzeniem jest ujemna wartość długości dotychczas przebytej drogi[2]. W ogólności, nie wszystkie akcje mogą być dopuszczalne w danym stanie. Przyjmuje się wtedy, że przy podjęciu takiej niedopuszczalnej akcji, stan nie zmienia się.

**Definicja 2.2.0.1** *Formalnie, proces decyzyjny składa się z:*

- *zbioru akcji $\mathcal{A}$ i stanów $\mathcal{S}$, z wyróżnionym stanem początkowym $s^0 \in \mathcal{S}$,*
- *rozkładu prawdopodobieństwa następnego stanu - $P(s_{t+1}|s_t, a_t)$, gdzie $s^t \in \mathcal{S}$, $a^t \in \mathcal{A}$, $t$ to dyskretny czas,*
- *rozkładu prawdopodobieństwa wartości wynagrodzenia - $P(r_t|s_t, a_t)$.*

Zbiór akcji można sobie wyobrazić jako zbiór możliwości agenta[3]. Stan oznacza stan środowiska agenta. Gdy agent porusza się lub w jakiś sposób zmienia swoją reprezentację, jego współrzędne i zmienne właściwości są częścią stanu środowiska. Wynagrodzenie, nawet jeśli w konkretnym modelu przydzielane jest przez agenta samemu sobie, uważa się za pochodzące od środowiska (bo zależy od jego stanu) lub od metaforycznego *krytyka*.

Każde przejście procesu decyzyjnego składa się z kroków w dyskretnym czasie. W kroku $t$ środowisko (wraz z agentem) jest w stanie $s_t$, agent podejmuje decyzję i wykonuje akcję $a_t$. Za tą decyzję otrzymuje wynagrodzenie $r_t$[4]. Zależnie od stanu poprzedniego i akcji agenta, środowisko zmienia stan $s_{t+1}$. Zmiana może nie być deterministyczna - podjęcie akcji $a$ w stanie $s$ nie musi całkowicie determinować nowego stanu. Zadaniem agenta jest podjęcie takiej akcji, by zmaksymalizować sumaryczną wartość oczekiwaną wynagrodzenia w przebiegu procesu[5].

---

[1]Pojęcie agenta będzie w tym rozdziale szeroko rozumiane. W abstrakcyjnym sensie, agent jest punktem odniesienia w przestrzeni w chwili $t$.

[2]Wynagrodzenie jest po to, by je maksymalizować, stąd ujemna wartość.

[3]W starym już dowcipie było - prawo, lewo, góra, dół i fajer!

[4]Stosuje się tu rozróżnienie ról - stan jest stanem środowiska, akcja należy do agenta, wynagrodzenie przychodzi od środowiska do agenta. To rozróżnienie nie musi odpowiadać podziałowi agent-środowisko w przykładach.

[5]Najczęściej maksymalizuje się wynagrodzenie dyskontowane w czasie, by faworyzować szybkie rozwiązania. O tym później.

Pozornie zdawałoby się, że wystarczy znaleźć dla każdego stanu $s \in \mathcal{S}$ odpowiadającą mu akcję $a \in \mathcal{A}$, maksymalizującą oczekiwaną jednokrokową wypłatę akcji dla stanu (metoda zachłanna):

$$r_s(a) = E_t(r^t|s^t = s, a^t = a) \tag{2.1}$$

Jednak strategia zachłanna nie zawsze jest optymalną - akcja dająca dobrą wypłatę może doprowadzić do stanu, dla którego żadna akcja nie da dobrego wynagrodzenia, a przejście do lepszego stanu będzie mało prawdopodobne lub niemożliwe - wydanie całych oszczędności jednego dnia daje chwilowe zadowolenie, ale prowadzi do natychmiastowego ubóstwa.

Należy więc szukać metody maksymalizującej sumaryczne wynagrodzenie do ustalonej chwili $t$ lub, w sytuacji gdy nie można wyróżnić końca, metody podejmowania decyzji maksymalizujących wynagrodzenie w pewnych ograniczonych odstępach czasu. Zwykle wyróżnia się tylko początek procesu i maksymalizuje się zdyskontowaną wartość oczekiwaną wynagrodzenia:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s\right] \tag{2.2}$$

gdzie $\gamma \in [0, 1]$ to współczynnik dyskontowania.

Parametr $\gamma = 0$ sprawia, że porównywane jest wyłącznie natychmiastowe wynagrodzenie, $\gamma = 1$ - suma wszystkich wzmocnień, która może być nieskończona w nieskończonym przebiegu. Rola współczynnika $\gamma$ będzie omówiona później (31).

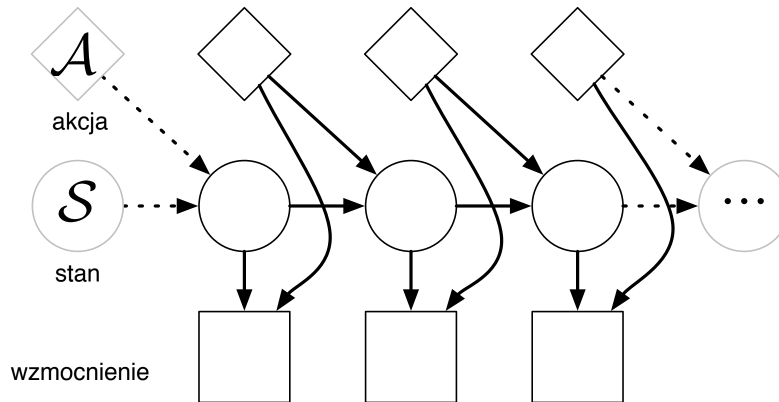
**Definicja 2.2.0.2** Procesem Markowa *nazywamy wektor zmiennych losowych* $\{X_t\}_{t\in\mathbb{N}}$, *spełniających własność:*

$$P(X_t = j|X_0 = i_0, X_1 = i_1, ..., X_{t-1} = i_{t-1}) = P(X_t = j|X_{t-1} = i_{t-1}) \tag{2.3}$$

czyli, interpretując $t$ jako czas, stan obecny jest warunkowo niezależny od przeszłości dalszej niż stan poprzedni. Proces Markowa można sobie wyobrazić jako graf skierowany, którego wierzchołkami są wektory wartości zmiennych losowych (stany), a krawędzie znaczone są prawdopodobieństwem przejścia między stanami.

Proces decyzyjny nazywamy procesem decyzyjnym Markowa (*Markov Decision Process, MDP*), o ile rozkład prawdopodobieństwa wartości wynagrodzenia i stanu następnego zależy wyłącznie od stanu obecnego i decyzji podjętej przez agenta - jest to proces Markowa, w którym znalazł się podejmujący decyzje agent. Jeśli ilość możliwych stanów i akcji jest skończona, mówimy o skończonym procesie decyzyjnym Markowa.

Gra w Samotnika jest skończonym procesem decyzyjnym Markowa - dokładny stan gry po wykonaniu ruchu, jak też możliwość pomyślnego zakończenia gry, deterministycznie zależy od wykonanego ruchu, ilość możliwych konfiguracji pionków jest skończona. Nieistotne, czy gracz zaczynał od stanu surowego, czy kontynuował czyjąś przerwaną grę




**Rysunek 2.1**. Proces decyzyjny Markowa

- jego decyzja w danym momencie będzie miała dokładnie taki sam skutek (rozkład skupiony w jednym punkcie). Mniej trywialnym przykładem jest ruletka - wybieram stawkę, decyduję się na liczbę i/lub kolor i, o ile kasyno jest uczciwe, tym samym determinuję rozkład prawdopodobieństwa możliwej wypłaty. Stan środowiska w czasie $t + 1$ również zależy od podjętej decyzji - choć w skutek moich działań koło nie zmieni swoich własności, jeśli pozbędę się żetonów, moje możliwości decyzyjne będą dość ograniczone.

Rozwiązywanie procesów decyzyjnych Markowa jest domeną uczenia ze wzmocnieniem.

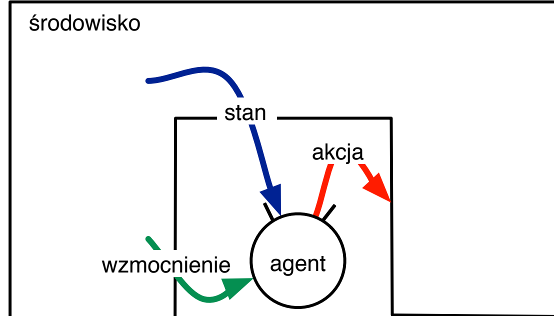
## 2.3. Uczenie ze wzmocnieniem

Uczenie ze wzmocnieniem (*reinforcement learning*) to motywowana psychologicznie technika uczenia maszynowego, w której systemowi uczącemu się nie jest podawana prawidłowa odpowiedź, a tylko wartościowanie każdego kroku, zwane wynagrodzeniem lub wzmocnieniem[6]. Zadanie agenta polega na wykształceniu strategii maksymalizującej oczekiwaną wartość wzmocnienia. Strategia może być deterministyczna: $\pi\colon \mathcal{S} \to \mathcal{A}$ lub niedeterministyczna: $\pi(a|s) = P(a_t = a|s_t = s), s \in \mathcal{S}, a \in \mathcal{A}$.

Uczenie ze wzmocnieniem nazywa się też uczeniem *z krytykiem*. Jest to bardzo obrazowa, ale o tyle niefortunna nazwa, że sugeruje możliwość krytyki jako formy karania agenta. Wzmocnienie może przyjmować dowolne wartości, ale zawsze liczy się porównanie - czy w stanie $s \in \mathcal{S}$ akcja $a \in \mathcal{A}$ przynosi większe wzmocnienie niż akcja $a' \in \mathcal{A}$. Nie wyróżnia się granicy, oddzielającej wzmocnienie nagradzające od karzącego i, choć model jest

[6]Termin wynagrodzenie (*reward*), którego używałem w poprzedniej sekcji, jest bardziej ogólny: termin wzmocnienie (*reinforcement*) zaczerpnięty jest z psychologii i zakłada zdolność agenta do uczenia się. Różnica jest w zasadzie formalna.

symetryczny i zawsze można postawić minus i minimalizować zamiast maksymalizować, przyjęło się, że wzmocnienie jest maksymalizowane.




**Rysunek 2.2**. Interakcja agenta ze środowiskiem w uczeniu ze wzmocnieniem

Chcąc porównać dwie strategie, $\pi_1$ i $\pi_2$, należy porównać nie tylko oczekiwaną wartość sumy wzmocnienia uzyskanego podczas eksploracji (w przypadku zapętleń wzmocnienie może być nieskończone), ale też uwzględnić czas uzyskiwania poszczególnych wartości wzmocnienia tak, aby „impuls" wzmocnienia, który nadszedł wcześniej był więcej wart niż taki sam impuls po większej ilości kroków. Czynnikiem odpowiadającym za dyskontowanie długotrwałych strategii jest wspomniany już czynnik $\gamma$.

**Definicja 2.3.0.3** *Funkcją jakości stanu $s \in \mathcal{S}$ dla strategii $\pi$ nazywamy funkcję wyznaczoną następująco:*

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s\right] \tag{2.4}$$

*gdzie $\gamma \in [0,1]$, a $\mathbb{E}_{\pi}$ oznacza wartość oczekiwaną przy stosowaniu strategii $\pi$.*

Funkcję $V$ można interpretować jako funkcję jakości strategii, przy ustalonym stanie początkowym lub wartości stanu $s \in \mathcal{S}$ dla strategii $\pi$.

Wyróżnia się też funkcję jakości pary stan-akcja dla strategii $\pi$.

**Definicja 2.3.0.4** *Funkcją jakości pary stan-akcja, lub $\mathcal{Q}$-funkcją nazywamy funkcję zadaną przez:*

$$\mathcal{Q}^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\hat{r}(s,a) + \sum_{t=1}^{\infty} \gamma^t r_t \middle| s_0 = s, a_0 = a\right] \tag{2.5}$$

*gdzie $\hat{r}(s,a)$ jest oczekiwanym wynagrodzeniem za podjęcie akcji $a$ w stanie $s$.*

Funkcja $\mathcal{Q}$ jest ściśle związana ze strategią $\pi$ - jej wartość dla danej pary stan-akcja zależy od tego, jaką strategię będzie wykonywał agent w dalszych krokach. Jednocześnie

parametry $a$ i $s$ są niezależne od strategii, dzięki czemu można podejść do problemu z drugiej strony - zdefiniujmy strategię $\pi'$ jako maksymalizację wartości funkcji $\mathcal{Q}^{\pi}$:

$$\pi'(s) = argmax_a \mathcal{Q}^{\pi}(s, a) \tag{2.6}$$

Agent wykonuje tą akcję w stanie $s$, dla której wartość $\mathcal{Q}^{\pi}(s, a)$ jest największa, przy założeniu, że w kolejnych krokach agent będzie wykonywał strategię $\pi$. Strategia $\pi'$ może być inna niż wyjściowa strategia $\pi$. Gdy $\pi$ nie wybiera optymalnej (w sensie $\mathcal{Q}$) akcji dla stanu $s$, strategia $\pi'$ wybiera bardziej opłacalną akcję, więc, przynajmniej w tym jednym kroku, daje większe wzmocnienie. Jednak stosowanie strategii $\pi'$ na dłuższą metę może stać się nieopłacalne - bazuje ona na założeniu, że kolejne akcje agent podejmuje według strategii $\pi$.

Powyższe zapętlające się rozumowanie prowadzi nas do szkicu iteracyjnej metody, w której każdy kolejny krok poprawia wydajność strategii. Dla strategii $\pi'$ istnieje odpowiadająca jej funkcja $\mathcal{Q}^{\pi'}$. Dla tej funkcji istnieje strategia $\pi''$, zdefiniowana analogicznie do $\pi'$, i tak dalej. Przy odpowiednich założeniach ten proces jest zbieżny. W tej szkicowej postaci algorytm jest niezwykle kosztowny w implementacji - przejście od $\pi$ do $\pi'$ wymaga nie tylko znalezienia maksimum wartości $\mathcal{Q}$ dla każdego stanu, ale też wyznaczenia nowej funkcji $\mathcal{Q}^{\pi'}$ przez rzeczywiste stosowanie strategii $\pi'$ przez pewien czas[7] dla każdego stanu początkowego i wszystkich możliwych akcji.

Przez $\mathcal{Q}$ oznacza się funkcję $\mathcal{Q}^{\pi}$ dla globalnie optymalnej strategii $\pi$[8]. W dalszej części rozdziału, $\mathcal{Q}$ może oznaczać pewną funkcję jakości pary stan-akcja, aproksymację takiej funkcji lub ciąg aproksymacji zbiegający do optymalnej funkcji $\mathcal{Q}$.

**Definicja 2.3.0.5** *Q-uczeniem nazywamy iteracyjną modyfikację wewnętrznej reprezentacji (tabelarycznej lub aproksymacyjnej) funkcji $\mathcal{Q}$ i odpowiadającej jej strategii $\pi$ tak, by z każdą modyfikacją zwiększać oczekiwaną, dyskontowaną wartość wynagrodzenia.*

## 2.4. Q-uczenie

Obserwacja o zależności funkcji $\mathcal{Q}$ od strategii sugeruje zastosowanie rekurencyjnego programowania dynamicznego. Skoro funkcja jakości pary stan-akcja wyznacza strategię, wystarczy trzymać w tabeli wartości funkcji $\mathcal{Q}$ dla poszczególnych stanów i akcji i wybierać w danym stanie akcję maksymalizującą wartość w wierszu[9], tzw. strategia zachłanna.

Wychodząc z obserwacji, że:

$$\mathcal{Q}(s_t, a_t) \approx r_{s_t}(a_t) + \gamma \cdot V(s_{t+1}) \tag{2.7}$$

[7] Tak, aby zmiana była obliczeniowo pomijalna dzięki czynnikowi dyskontującemu.

[8] Cały czas optymalność strategii zależy od czynnika $\gamma$. Zakładamy, że czynnik $\gamma$ jest ustalony w zadaniu.

[9] Przyjmijmy dla ustalenia uwagi, że w tabeli wiersz odpowiada stanowi, a kolumna - akcji.

a jednocześnie, przy stosowaniu strategii $\pi^{\mathcal{Q}}$:

$$V(s_{t+1}) \approx max_{a \in \mathcal{A}} \mathcal{Q}(s_{t+1}, a). \tag{2.8}$$

Stąd mamy sugestię algorytmu uaktualniania aproksymatora funkcji $\mathcal{Q}$:

$$\mathcal{Q}(s_t, a_t) := r_{s_t}(a_t) + \gamma max_{a \in \mathcal{A}} \mathcal{Q}(x_{t+1}, a). \tag{2.9}$$

Udowodniono ([21]), że dla skończonych procesów decyzyjnych Markowa taka tabelaryczna reprezentacja funkcji $\mathcal{Q}$ jest zbieżna, o ile mamy dane wypłaty dla wszystkich stanów i akcji (uczenie batchowe). Wystarczy zainicjować tabelę, by zawierała jednokrokowe oczekiwane wzmocnienia i uaktualniać synchronicznie.

W wersji on-line[15] algorytmu zamiast oczekiwanej wypłaty $r_{s_t}(a_t)$ używa się ostatnio uzyskanej wypłaty - $r_t$. W przypadku deterministycznym $r_t = r_{s_t}(a_t)$, w przypadku niedeterministycznym dłuższa eksploracja daje dobre przybliżenie wartości oczekiwanej.

W uczeniu on-line, kolejne komórki uaktualnia się asynchronicznie, w kolejności eksploracji - po $(s_t, a_t)$ uaktualniona zostanie $(s_{t+1}, a_{t+1})$. Algorytm aktualizacji 2.9 w tej wersji ma postać:

$$\mathcal{Q}(s_t, a_t) := (1 - \kappa)\mathcal{Q}(s_t, a_t) + \kappa \left[r_t + \gamma \cdot \mathcal{Q}(s_{t+1}, a_{t+1})\right] \tag{2.10}$$

i nosi nazwę *SARSA*, od piątki parametrów potrzebnych do dokonania aktualizacji: $s_t$, $a_t$, $r_t$, $s_{t+1}$ oraz $a_{t+1}$.

Z implementacyjnego punktu widzenia, agent dysponuje tabelą wartości funkcji $Q$ dla wszystkich możliwych stanów i akcji (lub sprytnym aproksymatorem), inicjalizowaną losowo lub heurystycznie[10]. Znajdując się w stanie $s_t \in \mathcal{S}$, agent znajduje w tabeli odpowiadający temu stanowi wiersz, w nim znajduje pole zawierające najwyższą wartość i wykonuje akcję odpowiadającą kolumnie, w której jest to pole. Następnie obserwuje wynagrodzenie $r_t$, nowy stan $s_{t+1}$, wykonuje akcję $a_{t+1}$ i aktualizuje pole $(s_t, a_t)$ w tabeli zgodnie z 2.9. Aktualizacja tabeli następuje zawsze po kolejnym kroku.

SARSA należy do klasy algorytmów rozwiązujących procesy decyzyjne Markowa na podstawie przesunięć czasowych (*time difference*, *TD-learning*).

Dla algorytmów klasy TD wyznacza się funkcję błędu czasowego (*TD-error*):

$$E_{TD}(s_t, a_t) = r_t + \gamma \mathcal{Q}(s_{t+1}, a_{t+1}) - \mathcal{Q}(s_t, a_t). \tag{2.11}$$

Dla SARSA błąd jest dokładnie różnicą między poprzednią, a nową wartością w komórce $(s_t, a_t)$ tabeli.

[10] Inicjalizacja powinna uwzględniać zakres wynagrodzenia dostępnego w problemie.

## 2.5. Częściowo obserwowalne procesy decyzyjne Markowa

Przykładem gry, która nie pasuje do definicji procesu decyzyjnego Markowa jest poker[11]. Każdy z graczy dysponuje tylko częścią opisu środowiska - widzi tylko swoje karty. Na podstawie informacji, że inni gracze są w podobnej sytuacji (też mają po 5 kart), gracz musi zmaksymalizować wypłatę, która zależy nie tylko od jakości posiadanych kart, ale też od kart posiadanych przez przeciwników (ta wiedza nie jest mu dana), jak też skłonności przeciwników do dorzucenia żetonów do puli. Działanie tylko na podstawie obserwowanej części stanu środowiska jest nieoptymalne - gracz, którego zakłady odzwierciedlają jakość posiadanego układu pięciu kart, co wydaje się być najlepszą strategią w obliczu posiadanych informacji, zdobędzie mniej żetonów niż gracz starający się odgadnąć jakość kart innych graczy (zmienne ukryte) i dopasuje swoje zakłady. Mając słabsze karty niż któryś z przeciwników można wygrać, o ile przeciwnik straci wiarę w swoje karty i je odłoży (blef). Gra jest tym bardziej ciekawa, gdy uczestniczy w niej kilku doświadczonych pokerzystów i każdy opracowuje swoją strategię na bazie założenia, że pozostali mogą prezentować pokerową twarz.

Proces decyzyjny Markowa nazywamy częściowo obserwowalnym (*Partially Observable Markov Decision Process, POMDP*) jeśli agentowi dana jest tylko fragmentaryczna informacja o stanie środowiska, zależna od stanu środowiska. Patrząc z punktu widzenia agenta - zakłada on, że znajduje się w procesie Markowa, ale nie zna pewnych czynników, które determinują przebieg procesu. Warto tu zauważyć, że te ukryte zmienne są zależne od akcji agenta - agent może nie znać skutków swoich decyzji. Gdyby znał te czynniki (karty przeciwników, mapę labiryntu), mógłby opracowywać na ich podstawie strategię maksymalizującą wypłatę (Q-uczenie).

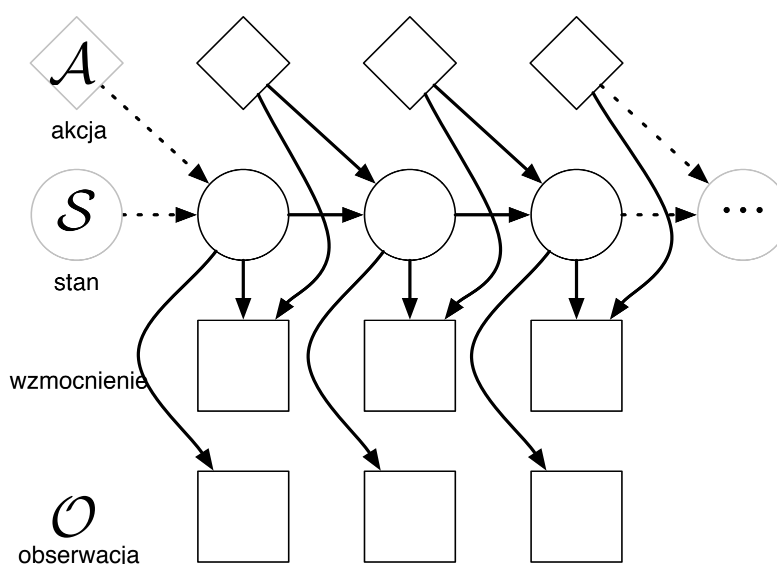
**Definicja 2.5.0.6** *Częściowo obserwowalny proces decyzyjny Markowa składa się z:*

- *zbioru stanów $\mathcal{S}$, akcji $\mathcal{A}$ i obserwacji $\mathcal{O}$,*
- *rozkładu stanów początkowych $\phi_0(s) = P(s_0 = s)$, $s \in \mathcal{S}$,*
- *rozkładu prawdopodobieństwa następnego stanu - $P(s_{t+1}|s_t, a_t)$, $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$,*
- *rozkładu prawdopodobieństwa wartości wynagrodzenia - $P(r_t|s_t, a_t)$, $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$,*
- *rozkładu prawdopodobieństwa obserwacji $P(o_t|s_t)$, $o_t \in \mathcal{O}$, $a_t \in \mathcal{A}$.*

Z definicji wynika, że nie tylko następny stan i wynagrodzenie są wartościami losowymi. Przy tym samym stanie środowiska, obserwacja poczyniona przez agenta nie musi

[11]Ponieważ bardziej doświadczeni gracze wygrywają częściej, poker w wielu krajach nie jest uważany za grę hazardową, a strategiczną lub nawet sport.

być zdeterminowana. Istoty biologiczne, w tym ludzie, zmuszone są stale uwzględniać tą niepewność obserwacji związaną z omylnością zmysłów. Iluzje optyczne pokazują nam jak bardzo jest to proces automatyczny. Widząc rysunek, który można dwuznacznie interpretować (słynny przykład - zając czy kaczka), jesteśmy w stanie „przestawić” swój umysł, skupiając się na cechach zajęczych lub kaczych rysunku. Brakujące elementy - w końcu to tylko kilka kresek, które interpretujemy jako reprezentację zwierzęcia - są stale dopisywane przez mózg, który działa zgodnie z założeniem, że po prostu nie zostały zaobserwowane. Gdyby na obecność określonych obiektów uaktywniałyby się wyspecjalizowane, nieomylne zmysły, być może nie wykształcilibyśmy myślenia abstrakcyjnego, symbolicznego, pisma, matematyki.




**Rysunek 2.3**. Częściowo obserwowalny proces decyzyjny Markowa

Terminologia przyjęta przy rozwiązywaniu częściowo obserwowalnych procesów decyzyjnych Markowa obejmuje pojęcie stanu przekonań (*belief state*). Jest to taki zbiór stanów i hipotetyczny rozkład prawdopodobieństwa obserwacji, dla którego proces, wraz z wcześniej dokonanymi obserwacjami, ma własność Markowa.

Rozmowa z chatterbotem opartym na języku AIML jest częściowo obserwowalnym procesem decyzyjnym Markowa, o ile skrypt nie korzysta z funkcji zapamiętywania pewnych (pasujących do ustalonego wzorca) elementów rozmowy. Język AIML (*Artificial Intelligence Markup Language*) umożliwia przygotowywanie gotowych zdań, będących odpowiedziami na zdania wpisywane przez rozmówcę. Boty w krótkiej[12] rozmowie mogą zdawać się przejawiać inteligencję. Skonfrontowane ze zdaniem, którego sensu nie mogą pojąć (patrz:

[12]Zależnie od obszerności skryptu i sprytu rozmówcy.

skrypt nie przewiduje), zaskakują nas nagle zmieniając temat lub opowiadając dowcip, ale natychmiast zdradzają swoją starannie skrywaną tajemnicę, nie mogąc odpowiedzieć na pytanie „Jakie było moje poprzednie pytanie?”. Jednak ich sukces jako wirtualnych sprzedawców polega właśnie na tym, że to w *naszej* naturze leży dopisywanie zmiennych ukrytych. Widząc uśmiechniętą twarz i litery pojawiające się na ekranie, jesteśmy skłonni do interakcji, zadawania pytań. Nawet świadomość, że twarz pochodzi z bazy danych, a wszystkie odpowiedzi sprzedawcy są uprzednio przygotowane, nie przeszkadza nam w traktowaniu go jako partnera do rozmowy. Iluzję podtrzymuje to, że baza danych bota ukierunkowana jest na jeden temat.

Proces decyzyjny nazywamy wektoryzowalnym (*factored MDP*) jeśli każdy stan $s \in \mathcal{S}$ i akcja $a \in \mathcal{A}$ (a w przypadku procesów częściowo obserwowalnych - również każda obserwacja) jest ustalonej długości wektorem zmiennych losowych. Gra w szachy daje się wektoryzować: stan szachownicy może mieć reprezentację wektora stanów pól lub wektora pozycji figur. Z naszych zmysłów, wektoryzować daje się na przykład wzrok lub słuch, ale nie węch. Doznania wzrokowe możemy opisywać niskopoziomowo, podając barwy, poziomy jasności i położenie, oprócz wyższego poziomu - interpretacji. Sygnał dźwiękowy, choć nie jesteśmy tego świadomi, jest w układzie słuchowym transformowany fourierowsko i analizowany. Nie potrafimy świadomie wyróżnić amplitud fal składowych, ale potrafimy podać ogólną charakterystykę transformaty - ton czysty, akord czy szum i określić kilka parametrów słyszanego dźwięku - głośność, wysokość. Węch opisujemy tylko przez określenie prawdopodobnego źródła zapachu na podstawie wcześniejszych doświadczeń, ewentualnie opisując siłę z jaką do nas dochodzi (kierunek już na podstawie dedukcji - wiatr, obserwacja wzrokowa). Między innymi dlatego dźwięk i obraz jesteśmy w stanie przesłać cyfrowym kanałem informacyjnym, a zapachu - nie[13].

Rozwiązywanie wektoryzowalnych, częściowo obserwowalnych procesów decyzyjnych Markowa jest wyzwaniem trudnym i ciekawym. Problemy wymagające znalezienia strategii maksymalizującej wzmocnienie mimo fragmentaryczności informacji o stanie otoczenia można mnożyć - pojawiają się zawsze tam, gdzie następuje interakcja ze światem, gdzie metaforyczny agent zmuszony jest opuścić bezpieczny świat matematycznych rozważań i wykazać się w rzeczywistości[14].

Pewne próby rozwiązywania POMDP podejmowano za pomocą sieci neuronowych, których rozliczne modele nierzadko korzystają z wewnętrznej reprezentacji przestrzeni danych w jednostkach ukrytych, by wykryć zachodzące w nich zależności.

[13]Również to, że nasz gatunek nie używa węchu w takim stopniu jak słuchu czy wzroku, ale kto by nie chciał pobrać z Internetu aromatu egzotycznych przypraw czy perfum?

[14]Naturalnie, nie każdy problem spotykany w rzeczywistości ma własność Markowa.

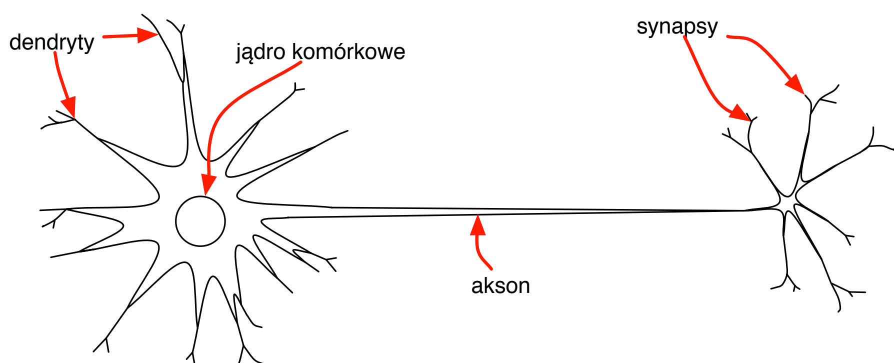
ROZDZIAŁ 3

# Maszyny Boltzmanna

Maszyny Boltzmanna są stosunkowo nową i szybko rozwijającą się klasą sieci neuronowych. Wprowadzone przez Geoffreya Hintona i Terry'ego Sejnowskiego w roku 1985 jako stochastyczna wersja sieci Hopfielda, były pierwszym modelem sieci neuronowych, w których uczeniu podlegały jednostki ukryte. Ceną za usprawnienia[1] była jednak ogromna kosztowność czasowa symulacji i uczenia. Do czasu, gdy w roku 2000 Hinton zaproponował niedokładną metodę uczenia, będącą przybliżeniem dokładnej metody, wymagającej próbkowania Monte-Carlo przestrzeni stanów.

## 3.1. Sieci neuronowe

Mózg ludzki to prawie półtorakilogramowe skupisko około stu miliardów komórek nerwowych[2]. Przez lata określany był przez popularyzatorów nauki jako „niezwykły zegar", „naturalny kalkulator", „superkomputer", a ostatnio nawet jako „wyszukiwarka internetowa", czyli zawsze jako najbardziej złożony instrument przetwarzający informację. Do dziś, mimo licznych odkryć neurofizjologicznych, i matematycznych modeli, nie został poznany do końca. Nadal nie mamy funkcjonalnego modelu zawartości naszych czaszek, ale z każdym krokiem przekonujemy się o nietrafności naszych wcześniejszych prób i porównań.




**Rysunek 3.1**. Neuron biologiczny, schemat

[1] Dla niecierpliwych - usprawnienia obejmują udowodnioną zbieżność do minimum globalnego z prawdopodobieństwem 1, brak oscylacji.

[2] Najbardziej śmiałe symulacje sztucznych sieci neuronowych ograniczone są do kilku tysięcy.

Sztuczne sieci neuronowe to niezwykle uproszczony matematyczny model naturalnych sieci, o funkcjonalności ukierunkowanej raczej na łatwość symulacji w postaci układów elektronicznych, czy programów komputerowych, niż na podobieństwo do rzeczywistych systemów biologicznych. W miarę rozwoju elektroniki i poszerzania się naszej wiedzy o naturalnych neuronach, sztuczne neurony zyskują na stopniu złożoności i możliwościach. Zdaje się, że obecnie największym problemem nie jest złożoność obliczeniowa potrzebna do choćby częściowej duplikacji funkcjonalności mózgu, a nasza niewiedza: w miarę wzrostu liczebności algorytmów i modeli, zdaje się wzrastać też liczba nierozwiązanych problemów. Jednocześnie badania behawioralne pokazują nam, że mózgi biologiczne z niezwykłą łatwością dokonują wyczynów, których nawet częściowo nie potrafimy zreplikować w maszynach. Obecnie, program komputerowy typu chatterbot uważa się za dobry, jeśli potrafi sprytnie udzielić odpowiedzi wymijającej na pytanie, którego treści nie zrozumiał[3]. Człowiek tymczasem, po krótkiej rozmowie z nowo poznaną osobą nie tylko jest w stanie zrozumieć każde zdanie, ale też na podstawie akcentu i doboru słów oszacować jej pochodzenie, wykształcenie, a nawet pewne cechy charakteru. To nie jest tylko kwestia większej bazy wiedzy o świecie, to niezwykła umiejętność obserwacji, uogólniania i filtrowania informacji płynącej z doznań zmysłowych pozwala nam funkcjonować w społeczeństwie i dynamicznie dopasowywać się do zmian w otoczeniu.

Jednak, nawet w prostych modelach, zauważane są pojedyncze cechy charakterystyczne dla naturalnych sieci neuronowych - zdolność adaptacji do nowych danych, generalizacji, charakterystyczne zachowanie przy stopniowym usuwaniu neuronów i przy przepełnieniu pojemności pamięciowej. Sztuczne sieci neuronowe nie są już akademicką zabawką, czy matematycznym kuriozum, ale znajdują zastosowanie w najróżniejszych dziedzinach, wymagających analizy danych, sterowania, przewidywania, czy przetwarzania sygnałów.

### 3.1.1. Neuron McCullocha-Pittsa

Sztuczne sieci neuronowe pojawiły się w świecie naukowym w czasach wczesnego rozwoju elektroniki, w roku 1943, gdy neurofizjolog Warren McCulloch i matematyk Walter Pitts opracowali matematyczny i elektryczny model naturalnego neuronu.

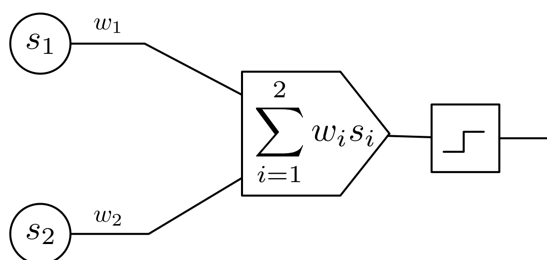
Wyjście neuronu progowego McCullocha-Pittsa wyznacza się w następujący sposób:

$$\begin{cases} 1; & \sum_{i=1}^{n} w_i s_i \geqslant \theta \\ 0; & \text{w.p.w.} \end{cases} \tag{3.1}$$

gdzie $n \in \mathbf{N}$ to ilość wejść, $s_i$, $i \in 1...n$ to wartości na wejściu (binarne, 0 lub 1[4]), a $w_i \in \{-1, 1\}$ to wagi przypisane wejściom.

[3] Najpierw musi potrafić oszacować, że poziom zrozumienia pytania jest poniżej pewnego poziomu.

[4] Tudzież bipolarne: -1 i 1, bez większej różnicy.




**Rysunek 3.2**. Schemat neuronu progowego

Taki prosty neuron jest w stanie klasyfikować podane na wejściu wzorce (wektory zero-jedynkowe za pomocą prostej reguły - jeśli ważona suma wejść nie przekracza ustalonego progu $\theta$ - wzorzec należy do pierwszej kategorii (0). Jeśli osiąga lub przekracza $\theta$ - do drugiej (1).

Struktura neuronu McCullocha i Pittsa była odzwierciedleniem ówczesnej fascynacji logiką formalną. Interpretując wartość 1 jako logiczną prawdę, a 0 jako fałsz, nietrudno przekonać się, że neuron jest w stanie odtworzyć niektóre operatory logiczne, między innymi - koniunkcję, przy $\theta = 2$ (i obu wagach ustalonych na 1), alternatywę, przy $\theta = 1$ ($w_1 = w_2 = 1$), zaprzeczenie logiczne, przy wadze na jedynym wyjściu równej $-1$ i progu $\theta = 0$. Niezdolność neuronu do otworzenia operatora alternatywy wyłącznej (*albo*, *wyłączne lub*, *exclusive or*, *XOR*) stała się głównym argumentem krytyki modelu i doprowadziła do spowolnienia pracy nad sztucznymi sieciami neuronowymi na długie lata.

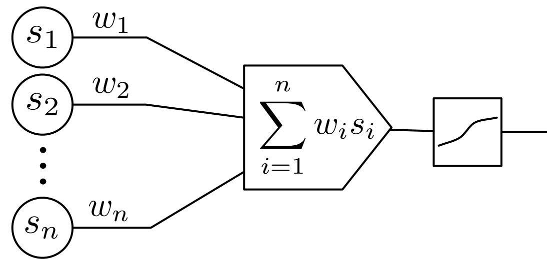
Problem XOR należy do szerokiej klasy problemów nie dających się separować liniowo: zagadnień, których nie da się rozwiązać przez podzielenie przestrzeni danych wejściowych hiperpłaszczyzną oddzielającą jedną kategorię od drugiej. Do takich zagadnień należy np. problem podzielności przez 3 (i dowolną inną liczbę oprócz potęg dwójki) liczb zadanych w systemie binarnym[5].

Rozwój wiedzy neurofizjologicznej doprowadził do kolejnych rewizji bądź co bądź - ograniczonego modelu. Reguła, sformułowana jeszcze w 1932 roku przez Donalda Hebba, będąca wnioskiem ze słynnych eksperymentów Pawłowa, otworzyła sieciom neuronowym nowe możliwości - zamiast ustalać wagi dla problemu, można uczyć neuron na podstawie dochodzących do niego sygnałów. Reguła Hebba mówi, że często używane synapsy są wzmacniane, a rzadko używane - osłabiane. Do reguły Hebba i uczenia hebbowskiego wrócimy później.

Wprowadzenie funkcji przejścia (*transfer function*) pozwoliło usprawnić proces uczenia się sieci. Motywacja jest jasna. Zadaniem neuronu jest dopasowanie nachylenia hiperpłaszczyzny w przestrzeni aktywacji wejść tak, by możliwie jak najlepiej oddzielała ona wejścia

[5]Operator XOR to szczególny przypadek problemu podzielności przez 3, dla liczb dwucyfrowych.

jednej kategorii od drugiej. W przypadku, gdy wzorce są klasyfikowane zero-jedynkowo, jakość klasyfikacji można mierzyć jedynie ilością źle sklasyfikowanych wzorców. Gdy neuron daje pewną miarę odległości od ustalonej granicy, można dokładniej zmierzyć jakość klasyfikacji. Choć nie zmienia to zdolności klasyfikacyjnych neuronu, przyspiesza uczenie i pozwala na generalizację: można się spodziewać, że neuron z odpowiednią funkcją przejścia uczony na pewnym podzbiorze zbioru danych lepiej klasyfikuje resztę zbioru niż neuron progowy uczony na tym samym podzbiorze. FIMXE progowa




**Rysunek 3.3**. Schemat neuronu o sigmoidalnej funkcji przejścia

Wyjście neuronu formalnego wyposażonego w funkcję przejścia oblicza się następująco:

$$s_i = \phi(\sum_{j=1}^{n} w_j s_j + \theta_i) \tag{3.2}$$

gdzie $\phi$ to ustalona funkcja przejścia, a $\theta_i$ to wartość równoważąca (ang. *bias*), charakterystyczna dla neuronu. W praktyce, wartości równoważące pomija się, a dodaje się do sieci dodatkowe wejście, stale równe $1$. Wagi między tym ustalonym wejściem, a neuronami podlegają procesowi uczenia.

Funkcje przejścia są najczęściej funkcje sigmoidalne: monotoniczne funkcje przyjmujące wartości $0$ i $1$ w $-\inf$ i $+\inf$ odpowiednio (lub $-1$ i $1$ w modelu bipolarnym). Typową funkcją sigmoidalną stosowaną w sieciach neuronowych jest *funkcja logistyczna*:

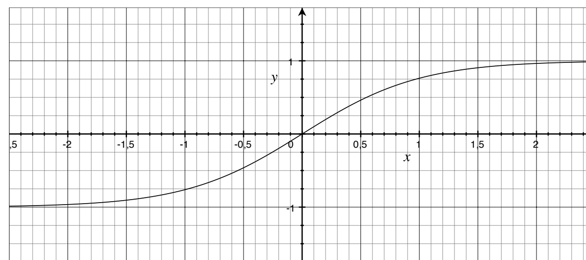
$$\sigma(x) = \frac{e^x}{1 + e^x} \tag{3.3}$$

oraz jej bipolarny odpowiednik - tangens hiperboliczny:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.4}$$

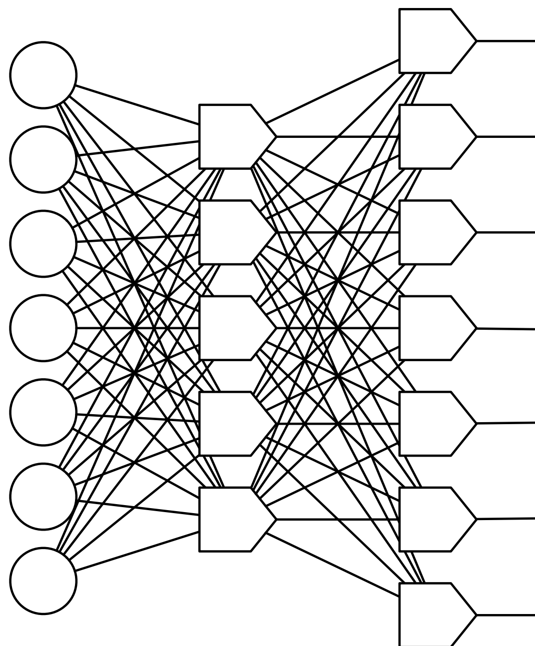
Odkrycie algorytmu propagacji wstecznej pozwoliło konstruować sieci wielowarstwowe, czyli takie, w których wejściem warstwy neuronów może być inna warstwa neuronów[6].

[6] Warstwy neuronów występują również w sieciach ontogenicznych, ale ten temat wybiega za daleko poza tematykę tej pracy.




**Rysunek 3.4**. Sigmoidalna funkcja używana często jako funkcja przejścia - tangens hiperboliczny.

Propagację wsteczną można z powodzeniem stosować do sieci o dowolnej architekturze, nawet o ścieżkach rekurencyjnych, o ile ustali się kolejność uaktualniania stanów neuronów. Algorytm propagacji wstecznej wymaga podania poprawnych odpowiedzi na zbiorze treningowym.



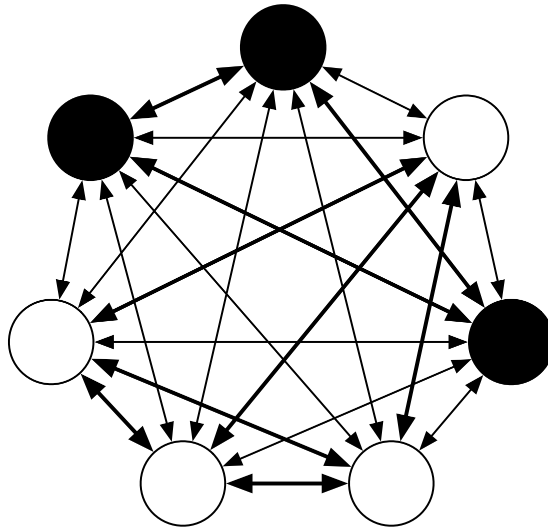
**Rysunek 3.5**. Perceptron wielowarstwowy o wielowymiarowym wyjściu

## 3.2. Sieci Hopfielda

W roku 1982 amerykański uczony John Hopfield zaproponował rekurencyjny model sieci o symetrycznych wagach i synchronicznej aktualizacji, z możliwością stosowania jako pamięć

autoasocjacyjna - przy wykorzystaniu reguły Hebba, tym samym odnawiając po dziesięciu latach zainteresowanie sztucznymi sieciami neuronowymi.

Podczas, gdy wcześniej rozważane sieci były sieciami jednokierunkowymi, model Hopfielda jest przykładem sieci w pełni rekurencyjnej. Każdy neuron może być w jednym z dwóch stanów: $-1$ lub $1$. Ponownie, model może być binarny lub bipolarny. Tym razem jednak, dla ustalenia uwagi, niech wszystkie neurony będą bipolarne.



**Rysunek 3.6**. Schemat sieci Hopfielda

Model zakłada połączenia między każdym neuronem, a każdym innym (brak połączeń zwrotnych):

$$W = [w_{ij}]_{i,j=1\ldots n}\,, w_{ii} = 0; \tag{3.5}$$

o wagach symetrycznych:

$$w_{ij} = w_{ji} \tag{3.6}$$

Czasem stosuje się neuron równoważący (*bias*), o stanie stałym i równym $1$. W dalszej części będę zakładał istnienie takiego neuronu w sieci.

Aktywacje neuronów obliczane są synchroniczne lub asynchronicznie. W wersji synchronicznej najpierw obliczane są wszystkie tzw. pola zewnętrzne:

$$a_i = \sum_{j=1}^{n} w_{ij} x_j \tag{3.7}$$

a następnie zmieniane są ich stany:

$$x_i = \begin{cases} 1; & a_i \geqslant 0 \\ -1; & a_i < 0 \end{cases} \tag{3.8}$$

Tu warto zauważyć, że obliczanie aktywacji w trybie synchronicznym równoważne jest mnożeniu wektora stanów przez macierz wag.

W trybie asynchronicznym aktywacje i stany neuronów uaktualnia się w ustalonym porządku lub losowo. Istnieją również modele obejmujące zjawisko *refrakcji* czyli nie dopuszczające, by neuron był aktywowany przez jakiś czas po ostatniej aktywacji. Refrakcja jest umotywowana biologicznie.

Dla danej konfiguracji stanów neuronów wyróżnia się *energię konfiguracji*:

$$E(x) = -\frac{1}{2}\sum_i\sum_j w_{ij}x_ix_j \tag{3.9}$$

Załóżmy, że zmieniony został stan neuronu $x_i$ na zgodny z polem wypadkowym: $x_i'$, czyli $x_ia_i \leqslant 0$ oraz $x_i'a_i \geqslant 0$. Wówczas:

$$E(x') - E(x) = -\frac{1}{2}\sum_j w_{ij}x_j(x_i' - x_i) = (x_i - x_i')a_i \leqslant 0 \tag{3.10}$$

Ponieważ ilość możliwych stanów sieci jest skończona, a energia stale maleje, sieć Hopfielda o dynamice asynchronicznej, zainicjowana w pewnym stanie, zatrzymuje (zapętla) się w stanie o lokalnie najniższej energii.

Stan na którym zatrzyma się sieć nie musi być minimum globalnym. Sieci Hopfielda utykają w minimach lokalnych.

## 3.3. Uczenie hebbowskie

Wspomnianą wcześniej regułę Hebba (synapsy częściej używane wzmacniają się, rzadziej używane - osłabiają) można interpretować w terminach korelacji:

$$\frac{dw_{ij}}{dt} \sim corr(s_i, s_j) \tag{3.11}$$

co oznacza, że zwiększać będziemy wagi tych połączeń, które łączą neurony o dodatnio skorelowanych stanach (po czasie), a zmniejszać tych, które łączą neurony o ujemnej korelacji. Jeśli stany neuronów są odpowiedziami na bodźce z otoczenia (jak to bywa w przypadku biotów), uczenie hebbowskie wykształca skojarzenie między bodźcami, które często występują razem: widząc ogień oczekujemy uczucia ciepła, widząc ciastko, przewidujemy jego słodki smak. Wagi odpowiadające za kojarzenie ognia z uczuciem wilgoci lub chłodu, a ciastka z kwaśnym lub gorzkim smakiem są w uczeniu hebbowskim osłabiane - nagłe pojawienie się dwóch nie pasujących do siebie bodźców wywołuje zaskoczenie.

Takie korelacyjne kojarzenie bodźców nazywa się autoasocjacją lub uzupełnianiem wzorca. Sieć Hopfielda uczona regułą Hebba funkcjonuje jako pamięć autoasocjacyjna - raz nauczona zestawu wzorców, po podaniu niepełnego lub zaszumionego wektora, odtworzy

wzorzec najbardziej podobny do danego. Różnica między niepełnym, a zaszumionym wzorcem jest w zasadzie formalna. O niepełnym mówimy, gdy pewien ustalony, ciągły fragment jest pozostawiony bez zmian, a reszta wzorca zastąpiona jest stałą wartością ($-1$ lub $1$), o zaszumionym - gdy losowo wybrane elementy wzorca zostały odwrócone.

Jak znaleźć macierz wag umożliwiającą wykorzystanie sieci hopfieldowskiej jako pamięci asocjacyjnej?

Dany mamy zbiór $P$ wzorców:

$$I^{\mu} = \{\xi_i\}^{\mu} \tag{3.12}$$

gdzie $i = 1, ..., N$, $N$ - wymiar wzorców, a $\mu$ indeksuje wzorce. Chcemy znaleźć funkcję energetyczną faworyzującą te wzorce jako minima energetyczne i na jej podstawie dobrać macierz wag.

Zdefiniujmy $M^{\mu}(x)$ jako miarę zgodności stanu sieci $x$ ze wzorcem $I^{\mu}$:

$$M^{\mu}(x) = \frac{1}{N}\sum_i x_j \xi_i^{\mu} = \frac{1}{N}\langle x, I^{\mu}\rangle \tag{3.13}$$

gdzie $\langle .,. \rangle$ oznacza iloczyn skalarny. W przypadku bipolarnym mamy do czynienia ze znormalizowaną odległością Hamminga.

Przyjmijmy:

$$E(x) = -\frac{1}{2}\sum_{\mu} N \cdot (M^m u(x))^2 \tag{3.14}$$

Wtedy:

$$E(x) = -\frac{1}{2N}\sum_{\mu}\sum_{i,j} x_i \xi_i^{\mu} \xi_j^{\mu} x_j - \frac{1}{2}P \tag{3.15}$$

Stałą $\frac{1}{2}P$ można pominąć. Przekształcając i przyrównując prawą stronę powyższego równania do energii konfiguracji:

$$-\frac{1}{2}\sum_{i,j}\left(\frac{1}{N}\sum_{\mu}\xi_i\xi_j\right)x_i x_j = -\frac{1}{2}\sum_{i,j} w_{i,j} x_i x_j \tag{3.16}$$

Uzyskujemy jawny przepis na macierz wag:

$$w_{i,j} = \eta \sum_{k=1}^{n} \xi_i^{\mu} \xi_j^{\mu} \tag{3.17}$$

W dodatku w pełni zgodny z regułą Hebba.

Należy tu zauważyć, że w proponowana funkcja energii jest symetryczna względem odwrotności wzorców - oprócz każdego zapamiętanego wzorca, atraktorem procesu przypominania będzie także jego negatyw.

Ponadto zauważmy, że sieć zapamiętuje w wagach statystykę drugiego rzędu dla wzorców, czyli jedynie korelacje między parami elementów, przez co mogą pojawiać się *wzorce pasożytnicze*, stanowiące w krajobrazie energetycznym lokalne minima. Przy większej

ilości wzorców do zapamiętania ilość takich wzorców pasożytniczych gwałtownie wzrasta, a różnica energii między właściwym, a pokrewnym mu pasożytniczym wzorcem - maleje. Ogólna stabilność pamięci asocjacyjnej zależy od korelacji między parami wzorców do zapamiętania: silnie skorelowane wzorce będą mylone między sobą.

Wprowadza się również ciągłą wersję sieci Hopfielda. Ciągła sieć Hopfielda ma identyczną budowę co wersja dyskretna, różni się tylko metodą obliczania stanu neuronu - zamiast funkcji progowej, dysponuje funkcją sigmoidalną:

$$a_i = \sum_{j=1}^{n} w_{ij} x_j \tag{3.18}$$

$$x_i = tgh(a_i \cdot b_i) \tag{3.19}$$

gdzie $b_i$ to współczynnik regulujący nachylenie środka sigmoidy.

## 3.4. Dyskretne maszyny Boltzmanna

Maszyny Boltzmanna to stochastyczna wersja sieci Hopfielda zaproponowana przez Hintona i Sejnowskiego w 1985. Modyfikacja polega na tym, że energia nie maleje z każdym krokiem symulacji, a może wzrastać, z prawdopodobieństwem wykładniczo zależnym od tego wzrostu.

W maszynie Boltzmanna aktywację (pole wypadkowe) oblicza się analogicznie do sieci Hopfielda:

$$a_i = \sum_{j=1}^{n} w_{ij} x_j \tag{3.20}$$

Zmiana stanu jest bardziej złożonym procesem - prawdopodobieństwo zmiany stanu neuronu zależy od energii nowej konfiguracji.

- losuj neuron do zmiany: $x_i$
- jeśli stan (spin) neuronu jest niezgodny z polem wypadkowym, zmień: $x_i := sgn(a_i)$
- jeśli stan neuronu jest zgodny z polem wypadkowym, zmień na przeciwny z prawdopodobieństwem $P = e^{-2\beta|a_i|} = e^{-\beta(E(x')-E(x))}$, z prawdopodobieństwem $1 - P$ nie zmieniaj.

gdzie konfiguracja $x'$ różni się od $x$ stanem $i$-tego neuronu.

Czynnik $\beta$ nazywa się też czasem temperaturą odwrotną ($\frac{1}{T}$). Im $\beta$ bliższa zeru, czyli wyższa temperatura, tym chaotyczniej zachowuje się sieć: Przy $\beta \to \infty$ maszyna Boltzmanna zachowuje się jak zwykła sieć Hopfielda z deterministycznymi uaktualnieniami.

Symulacja maszyny Boltzmanna w trybie przywoływania wzorca polega na wielokrotnym losowaniu neuronu do zmiany. Każda zmiana zwiększa lub zmniejsza energię, przy czym wartość oczekiwana zmiany energii jest w czasie przebiegu sieci najczęściej ujemna - stany neuronów stopniowo dopasowują się do wag, następuje tak zwana *relaksacja* sieci. Ile takich zmian pojedynczych neuronów należy wykonać, aby sieć znalazła się w stanie stacjonarnym - nie jest jednoznacznie określone. Istnieją algorytmy gwarantujące[7], że sieć znajduje się w stanie stacjonarnym, ale ich złożoność obliczeniowa jest zbyt duża, by stosować je w rzeczywistej symulacji.

Podstawową zaletą maszyn Boltzmanna jest więc to, że mogą opuszczać stany będące minimami jedynie lokalnymi. Ewolucja sieci Hopfielda jest wędrówką według gradientu po krajobrazie energetycznym - jak spływająca górska woda, która skupia się w jeziorach. Maszyna Boltzmanna ewoluuje jak niestrudzony turysta górski, który chce wejść na sam szczyt, więc raz na jakiś czas schodzi niżej, nie zadowalając się lokalnymi wzniesieniami[8].

Warto zauważyć, że w procesie minimalizacji energii bardzo niepożądanym zjawiskiem jest nagły skok do stanu o wyższej energii, gdy już było „dość dobrze". Tak jak na początku wędrówki dopuszczalne jest chaotyczne zachowanie, które może pozwolić znaleźć odpowiedni obszar przestrzeni energetycznej (taki o dużym spadku, który sugeruje bliskość minimum globalnego), tak w okolicach globalnego minimum nie opłaca się już skakanie do wyższych terenów, bo opóźnia to tylko osiągnięcie owego minimum. Dla maszyn Boltzmanna stosuje się dwa popularne algorytmy regulacji temperatury - symulowane wyżarzanie (*simulated annealing*) i symulowane studzenie (*simulated tempering*).

Symulowane wyżarzanie opiera się na założeniu, że z kolejnymi krokami symulacji sieć zbliża się do minimum energetycznego. Temperaturę (tudzież $\beta$) uzależnia się od pewnej arbitralnie ustalonej funkcji malejącej (rosnącej) z kolejnymi krokami symulacji tak, by chaotyczne zachowanie sieci malało z czasem.

W algorytmie typu simulated tempering[9], oprócz prawdopodobieństwa zmiany stanu, wyróżnia się też prawdopodobieństwo zmiany temperatury: losuje się kandydata na nową temperaturę (odwrotną) $\beta'$ i określa się jakość zmiany. Jakoś zmiany jest proporcjonalna do wielkości zmiany i prawdopodobieństwa wystąpienia aktualnego stanu w nowej temperaturze. To wyznacza prawdopodobieństwo zmiany temperatury na $\beta'$.

[7] Algorytm Proppa-Wilsona.

[8] To poetyckie porównanie wymaga odwrócenia kierunku - szczyt odpowiada minimum energii.

[9] Unikam tłumaczenia „studzenie".

## 3.5. Uczenie maszyn Boltzmanna

Cel uczenia maszyny Boltzmanna jest identyczny jak w przypadku autoasocjacyjnej sieci Hopfielda - mamy pewien zbiór danych, chcemy znaleźć takie wagi, dzięki którym sieć będzie pełniła funkcję pamięci autoasocjacyjnej.

Niech $q^0$ oznacza rozkład empiryczny po zbiorze danych, a $q^\infty$ rozkład stacjonarny w maszynie Boltzmanna, zlaeżny jedynie od wag i $\beta$. Naszym celem jest znalezienie takiego zestawu wag, który minimalizuje rozbieżność Kullbacka-Liebera tych dwóch rozkładów:

$$\begin{aligned} H(q^0|q^\infty) &= \sum_x q^0(x) ln\left(\frac{q^0(x)}{q^\infty(x)}\right) \\ &= \sum_x q^0(x)\log(q^0(x)) -' sum_x q^0(x)\log(q^\infty(x)) \end{aligned} \tag{3.21}$$

Mamy:

$$\frac{\partial H(q^0|q^\infty)}{\partial w_{ij}} = -\sum_x \frac{q^0(x)}{q^\infty(x)} \cdot \frac{\partial q^\infty(x)}{\partial w_{ij}} \tag{3.22}$$

Rozkład stacjonarny $q^\infty$ zależy wyłącznie od wag i temperatury:

$$q^\infty(x) = \frac{exp(-\beta E(x))}{Z[\beta]} \tag{3.23}$$

gdzie $Z[\beta] = \sum_c exp(-\beta E(c))$, $c$ indeksuje całą przestrzeń.

Pochodna licznika:

$$\frac{\partial exp(-\beta E(x))}{\partial w_{i,j}} = -\beta exp(-\beta E(x)) x_i x_j \tag{3.24}$$

Pochodna mianownika:

$$\frac{\partial Z[\beta]}{\partial w_{i,j}} = -\sum_c \beta exp(-\beta E(c)) c_i c_j \tag{3.25}$$

Ze wzoru na pochodną ilorazu:

$$\begin{aligned} \frac{\partial q^\infty(x)}{\partial w_{i,j}} &= \frac{Z[\beta](-\beta exp(-\beta E(x))x_i x_j) + exp(-\beta E(x))(\sum_c \beta exp(-\beta E(c))c_i c_j)}{Z[\beta]^2} = \\ &= -\beta \frac{exp(-\beta E(x)}{Z[\beta]} x_i x_j + \beta \frac{\sum_c exp(-\beta E(c))}{Z[\beta]} \cdot \frac{exp(-\beta E(x)}{Z[\beta]} = \\ &= -\beta q^\infty(x) x_i x_j + \beta < x_i x_j >_{q^\infty} \cdot q^\infty(x) \end{aligned} \tag{3.26}$$

Więc:

$$\begin{aligned} \frac{\partial H(q^0|q^\infty)}{\partial w_{i,j}} &= -\sum_x \left[\frac{q^0(x)}{q^\infty(x)} \left(\beta x_i x_j \cdot q^\infty(x) + < x_i x_j >_{q^\infty} \cdot q^\infty(x)\right)\right] = \\ &= \sum_x -\beta x_i x_j q^0(x) - \sum_x \langle x_i x_j \rangle_q^\infty q^0(x) = \\ &= \beta \cdot \mathbb{E}_{q^0}\left(x_i x_j - \langle x_i x_j \rangle_q^\infty\right) = \\ &= \beta\left(\langle x_i x_j \rangle_{q^0} \langle x_i x_j \rangle_{q^\infty} -\right) \end{aligned} \tag{3.27}$$

Co daje następującą regułę na zmianę wag:

$$\Delta w_{ij} = \eta \cdot \left( \langle x_i x_j \rangle_{q^0} - \langle x_i x_j \rangle_{q^\infty} \right) \tag{3.28}$$

gdzie $\eta$ to stała uczenia.

W praktyce, korelacje po danych oblicza się jako średnią po wzorcach, obliczanie korelacji w stanie stacjonarnym przyspiesza się przez relaksację sieci na próbce Monte Carlo przestrzeni danych.

Maszyny Boltzmanna mają jeszcze jedną innowacyjną cechę - dopuszczają uczenie jednostek ukrytych. Neuron uważamy za ukryty jeśli jego stan nie jest brany pod uwagę jako część zapamiętanego wzorca w procesie uczenia. Jednostki ukryte pozwalają zwiększyć moc obliczeniową sieci, uwalniając ją od statystyki pierwszego rzędu i zapamiętywania wzorców wyłącznie na podstawie korelacji między parami elementów.

Maszynę z neuronami ukrytymi uczy się według tej samej reguły zmiany wag, z tym, że tam, gdzie chcemy policzyć korelacje dla jednostek ukrytych, stosuje się relaksację przy zamrożonych jednostkach widzialnych.

### 3.5.1. Minimalizacja rozbieżności kontrastywnej

Publikacja Hintona „Training Products of Experts by Minimizing Contrastive Divergence"[6] z roku 2000 przyniosła rewolucję w uczeniu maszyn Boltzmanna. Hinton zauważył, że czynnikiem obniżającym wydajność modeli składających się z połączenia wielu części, uczonych na tych samych danych (ekspertów) jest to, że ich „zakresy kompetencji" nachodzą na siebie. W perceptronie, w którym każdy neuron odpowiada za dokładnie jedną kategorię nie ma tego problemu: neurony są predestynowane do swoich kategorii i zgodnie z tym założeniem uczone. W przypadku pamięci autoasocjacyjnej nie mamy jednak dobrze zdefiniowanych kategorii. Mówiąc metaforycznie - nie jesteśmy w stanie stwierdzić, że najoptymalniejszą strategią zapamiętywania twarzy jest np. skupianie się na nosie i ustach. Trenując model oparty na kilku niezależnie uczonych ekspertach należy zadbać, aby nie wchodzili sobie w drogę - jeśli jeden skupia się na kształcie oczu, inni niech skupiają się na nosie, podbródku czy brwiach. Algorytm uczenia powinien zapewniać, aby w wykształcającej się wewnętrznej reprezentacji problemu, zakresy kompetencji ekspertów obejmowały niezależne wymiary. Stąd nazwa - zasada minimalizacji rozbieżności kontrastywnej (*MCD rule*).

Przez słowo *ekspert* rozumieć będziemy model podejmujący decyzje na podstawie wektorów danych, przysłowiową czarną skrzynkę, dysponującą wektorem parametrów $\theta$, zmiennych w procesie uczenia. Metoda minimalizacji kontrastywnej rozbieżności wymaga od eksperta jedynie, aby dawało się obliczyć prawdopodobieństwo warunkowe wektora danych przy zadanych parametrach: $p(d|\theta)$. Eksperta będziemy utożsamiać z wektorem jego parametrów.

Prawdopodobieństwo danych $d$ dla połączenia ekspertów $\theta_1, ..., \theta_n$ jest unormowanym iloczynem prawdopodobieństw dla poszczególnych ekspertów:

$$p(d|\theta_1, ..., \theta_n) = \frac{\prod_m p_m(d|\theta_m)}{\sum_c \prod_m p_m(c|\theta_m)} \tag{3.29}$$

gdzie $c$ indeksuje wszystkie wektory w przestrzeni danych. Dla przestrzeni ciągłej, sumę zastępuje się całką po całej przestrzeni.

Uczenie zespołu ekspertów $\theta_1, ..., \theta_n$ zbioru danych $d$ wiąże się wtedy z minimalizacją logarytmu powyższego prawdopodobieństwa dla każdego eksperta ($\theta_m$) zgodnie z zespołem:

$$\frac{\partial \log p(d|\theta_1 ... \theta_n)}{\partial \theta_m} = \frac{\partial \log p_m(d|\theta_m)}{\partial \theta_m} - \sum_c p(c|\theta_1 ... \theta_n) \frac{\partial \log p_m(c|\theta_m)}{\partial \theta_m} \tag{3.30}$$

Ostatni element powyższego równania jest najbardziej kosztowny obliczeniowo.

W przypadku maszyn Boltzmanna, połączeniem niezależnych ekspertów jest tzw. *ograniczona maszyna Boltzmanna* czyli sieć o dynamice maszyny Boltzmanna, ale o niepełnym zbiorze synaps. W ograniczonej maszynie Boltzmanna (*Restricted Boltzmann Machine, RBM*) istnieją połączenia tylko między neuronami ukrytymi, a widzialnymi - nie ma połączeń wewnątrz warstwy ukrytej, ani wewnątrz widzialnej. Parametrami $\theta_i$ są wagi między warstwą widzialną, a $i$-tym neuronem warstwy ukrytej.

Hinton proponuje minimalizować nie tyle rozbieżność Kullbacka-Liebera między rozkładem po danych ($q^0$), a rozkładem w stanie stacjonarnym ($q^\infty$), co jest niezwykle kosztowne obliczeniowo, ale różnicę tej rozbieżności i rozbieżności pomiędzy $q^0$ a $q^1$, gdzie $q^1$ oznacza rozkład po pewnym *kroku* symulacji[10]. Hinton argumentuje, że skoro $q^1$ jest bliższe stanu stabilnego niż $q^0$, rozbieżność kontrastywna jest zawsze nieujemna:

$$H(q^0|q^\infty) - H(q^1|q^\infty) \geqslant 0 \tag{3.31}$$

Przy czym zero osiągane jest dokładnie wtedy, gdy podczas kroku symulacji nie zaszła żadna zmiana w rozkładzie (czyli - sieć już była w stanie stacjonarnym).

Z obliczeniowego punktu widzenia minimalizacja rozbieżności kontrastywnej zamiast rozbieżności Kullbacka-Liebera oznacza redukcję kosztownego obliczeniowo składnika z równania 3.30:

$$-\frac{\partial}{\partial \theta_m}(H(q^0|q^\infty) - H(q^1|q^\infty)) = \left\langle \frac{\partial \log p_m(d|\theta_m)}{\partial \theta_m} \right\rangle_{q^0} - \left\langle \frac{\partial \log p_m(d|\theta_m)}{\partial \theta_m} \right\rangle_{q^1} + \frac{\partial q^1}{\partial \theta_m} \frac{\partial H(q^1|q^\infty)}{\partial q^1} \tag{3.32}$$

Z testów przeprowadzonych przez Hintona[6] wynika, że wystarczy minimalizować różnicę pierwszych dwóch składników prawej strony powyższego równania. Trzeci składnik okazuje się być mały w proporcji do dwóch pierwszych i może być pominięty.

[10]Krok nie oznacza tu zmiany pojedynczego neuronu, ale ustalony arbitralnie czas symulacji. Im dłuższy, tym lepiej.

Daje to następującą regułę:

$$\Delta\theta_m \propto \left\langle \frac{\partial \log p_m(d|\theta_m)}{\partial \theta_m} \right\rangle_{q^0} - \left\langle \frac{\partial \log p_m(d|\theta_m)}{\partial \theta_m} \right\rangle_{q^1}. \tag{3.33}$$

W przypadku $j$-ego eksperta wewnątrz maszyny Boltzmanna, mamy zależność:

$$\frac{\partial logp_j(d|w_j)}{\partial w_{i,j}} = \langle x_i x_j \rangle_d - \langle x_i x_j \rangle_{q^\infty(j)} \tag{3.34}$$

gdzie $w_j$ to wagi między $j$-ym neuronem ukrytym a neuronami widzialnymi, $\langle x_i x_j \rangle_d$ to wartość oczekiwana korelacji przy $d$ zamrożonym w neuronach ukrytych, a $\langle x_i x_j \rangle_{q^\infty(j)}$ to korelacje obliczone podczas symulacji polegającej na naprzemiennym symulowaniu warstwy widzialnej przy niewidzialnej zamrożonej i niewidzialnej przy zamrożonej widzialnej.

Dla ograniczonej maszyny Boltzmanna minimalizacja rozbieżności kontrastywnej wygląda i oblicza się niezwykle prosto:

$$-\frac{\partial}{\partial \theta_m}(H(q^0|q^\infty) - H(q^1|q^\infty)) = \langle x_i x_j \rangle_{q^0} - \langle x_i x_j \rangle_{q^1} \tag{3.35}$$

Jako jeden krok symulacji (aby znaleźć rozkład $q^1$), Hinton proponuje następującą procedurę:

1. wybierz wektor danych z $d$,
2. ustal stany neuronów widocznych na ten wektor
3. symuluj warstwę ukrytą, przy zamrożonej widocznej
4. przy zamrożonej warstwie ukrytej symuluj widoczną

W przypadku ograniczonej maszyny Boltzmanna, brak połączeń wewnątrz warstw sprawia, że w każdym kroku wszystkie połączenia są między jednostkami zamrożonymi, a symulowanymi. Kilkukrotne wylosowanie tego samego neuronu do zmiany może jedynie zwiększyć prawdopodobieństwo jego zmiany. Wystarczy więc bardzo krótka symulacja każdej z warstw - tak, by każdy neuron miał po kilka szans zmiany stanu.

## 3.6. Ograniczone maszyny Boltzmanna na modelu ciągłym

Modyfikacja maszyn Boltzmanna na model ciągły jest analogiczna do modyfikacji sieci Hopfielda: przy tej samej dynamice zastępuje się progową funkcję przejścia przez sigmoidalną, z dodanym szumem gaussowskim:

$$a_j = \sigma_j(\sum_i w_{ij} x_i + \frac{1}{\beta} \cdot N(0,1)), \tag{3.36}$$

gdzie:

$$\sigma_j(x) = \frac{1}{1 + exp(-b_j x)} \tag{3.37}$$

gdzie $b_j$ jest, podobnie jak w przypadku sieci Hopfielda, czynnikiem regulującym stromość środka sigmoidy.

Energia konfiguracji wyraża się podobnie do energii konfiguracji w modelu ciągłym Hopfielda:

$$E(x) = -\frac{1}{2}\sum_{i,j} w_{i,j}x_i x_j + \sum_i \frac{\rho_i}{b_i}\int_0^{x_i} \sigma^{-1}(s)ds \tag{3.38}$$

gdzie $\rho_i$ to indywidualny parametr oporności neuronu.

Ponieważ ograniczona maszyna Boltzmanna na modelu ciągłym jest iloczynem ekspertów, z energii mamy regułę uczenia MCD:

$$\Delta w_{i,j} = \eta_w(\langle x_i x_j\rangle_{q^0} - \langle x_i x_j\rangle_{q^1}) \tag{3.39}$$

dla wag oraz:

$$\Delta b_{i,j} = \eta_b \left(\frac{\rho_j}{b_j^2}\left\langle \int_{x'_j}^{x_j} \sigma^{-1}(s)ds \right\rangle_{q^0}\right) \tag{3.40}$$

gdzie $x'_j$ to stan $j$-ego neuronu po jednokrokowej[11] symulacji.

Chen i Murray w [2] proponują uprościć 3.40 stosując przybliżenie:

$$\int_{x'_j}^{x_j} \sigma^{-1}(s)ds \propto (x_j + x'_j)(x_j - x'_j) \tag{3.41}$$

Uczona regułą MCD z proponowanymi przybliżeniami ciągła, ograniczona maszyna Boltzmanna daje się łatwo implementować jako układ scalony.

## 3.7. Ograniczone maszyny Boltzmanna a procesy decyzyjne Markowa

Brian Sallans, przy współpracy z Hintonem, wydał serię publikacji rozważających zastosowanie maszyn Boltzmanna do rozwiązywania wektoryzowalnych procesów decyzyjnych Markowa ([17], [18]).

Przypominając, proces decyzyjny Markowa to proces Markowa uzupełniony o pewną mniej lub bardziej zdeterminowaną wypłatę, zależną od stanu i akcji. Metaforycznego wykonawcę akcji nazywamy *agentem*, zmieniany wskutek akcji stan utożsamiamy ze *środowiskiem*, a mechanizm przydzielania wypłat określany jest jako *krytyk*. Ponadto, gdy agent posiada zdolność uczenia, wypłata nazywana jest *wzmocnieniem*. Rolą uczącego się agenta jest maksymalizacja wartości oczekiwanej zdyskontowanego wzmocnienia po wszystkich

[11]Ponieważ sieć jest ograniczona, stosuje się procedurę jak na str. 50.

możliwych przejściach procesu decyzyjnego, z uwzględnieniem prawdopodobieństwa ich wystąpienia (zmiana stanu w sposób losowy zależy od akcji i stanu poprzedniego):

$$\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r_t \middle| s_0 = s\right] \tag{3.42}$$

W ogólności, przez przejście procesu decyzyjnego rozumie się dowolny ciąg par stan-akcja $(s, a) \in \mathcal{S} \times \mathcal{A}$, jednak dla agenta uczącego się, wyróżnia się strategię, $\pi(s)$ lub $\pi(a|s)$, na podstawie której agent podejmuje decyzję o akcji w danym stanie.

Dla strategii $\pi$ wyznacza się jakość pary stan-akcja:

$$\mathcal{Q}^{\pi}(s, a) = \mathbb{E}_{\pi}\left[r_s(a) + \sum_{t=1}^{\infty}\gamma^t r_t \middle| s_0 = s, a_0 = a\right] \tag{3.43}$$

która sama w sobie wyznacza nową strategię, polegającą na wybieraniu tej akcji, która maksymalizuje $\mathcal{Q}$ dla danego stanu $s \in \mathcal{S}$, zgodnie z rekurencyjnym równaniem:

$$\pi^{\mathcal{Q}}(s) = argmax_a \mathcal{Q}^{\pi}(s, a) \tag{3.44}$$

Zaczynając od tabeli wzmocnień za pary stan-akcja (lub od losowych wartości), stabelaryzowana lub aproksymowana funkcja $\mathcal{Q}$ zbiega do wyznaczającej optymalną strategię, gdy jest aktualizowana synchronicznie, algorytmem programowania dynamicznego lub asynchronicznie, algorytmem SARSA lub innym, klasy TD.

W przypadku gdy akcji jest dużo, znajdowanie takiej akcji na podstawie aproksymowanej lub stabelaryzowanej $\mathcal{Q}$-funkcji może być kosztowne obliczeniowo lub pamięciowo. Przy dużej ilości możliwych stanów tabelaryzacja funkcji $\mathcal{Q}$ może w ogóle okazać się nieopłacalna - to tak jakby tabelaryzować wszystkie możliwe stany szachownicy po to, żeby znaleźć akcję zwiększającą prawdopodobieństwo wygranej. Tabelaryzacja jest nieopłacalna również z tego względu, że niektóre stany są bardziej prawdopodobne niż inne i agent mógłby popaść w pułapkę korzystania zaledwie z niewielkiej części ogromnej bazy danych wynagrodzeń, nie mogąc przy krótkiej eksploracji określić, które stany może z bazy usunąć.

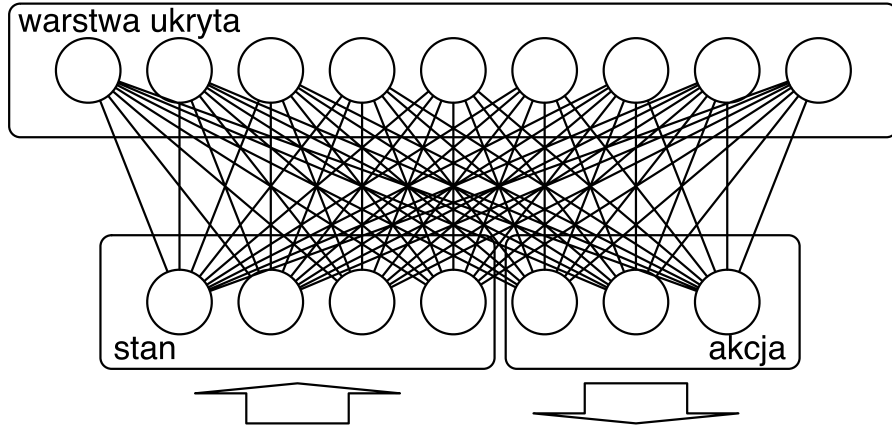
Sallans i Hinton w [17] proponują wykorzystać ograniczone maszyny Boltzmanna do aproksymacji niedeterministycznej strategii $\pi^{\mathcal{Q}}(a|s) = P(a|s)$.

Oznaczmy przez $v$ warstwę widoczną, a przez $h$ - ukrytą. Ponadto, niech $v$ dzieli się na dwa rozłączne podzbiory - $s$ i $a$, jak na rys. 3.7.

Energia konfiguracji $(v, h)$ przedstawia się przy tych oznaczeniach następująco:

$$E(v, h) = -\sum_{i,k} w_{ik} v_i h_k - \sum_{i<j} w_{ij} v_i v_j - \sum_{k<m} w_{km} h_k h_m \tag{3.45}$$

gdzie $i$ oraz $j$ indeksują jednostki widzialne, a $k$ i $m$ - ukryte.




**Rysunek 3.7**. Schemat ograniczonej maszyny Boltzmanna realizującej pary stan-akcja

Energia determinuje rozkład prawdopodobieństwa konfiguracji $(v, h)$ zgodnie z rozkładem Boltzmanna:

$$P(v,h) = \frac{exp(-E(v,h))}{\sum_{\hat{v},\hat{h}} exp(-E(\hat{v},\hat{h}))} \tag{3.46}$$

gdzie $\hat{v}$ i $\hat{h}$ to wszystkie możliwe konfiguracje.

Jakie jest więc prawdopodobieństwo wystąpienia konkretnego stanu jednostek widocznych $v$ w stanie stacjonarnym?

Przez $F_q(v)$ oznaczmy wolną energię wariacyjną:

$$F_q(v) = \sum_h q(h)E(v,h) + \sum_h q(h)logq(h) \tag{3.47}$$

gdzie $q$ to dowolny rozkład po konfiguracjach zmiennych ukrytych $h$.

Minimalizacja $F_q(v)$ wymaga, aby rozkład $q$ skupiał się na tych konfiguracjach zmiennych ukrytych, dla których energia konfiguracji jest niska, a jednocześnie maksymalizował entropię. Rozkład Boltzmanna spełnia te wymogi.

$$P(h|v) = \frac{exp(-E(v,h))}{\sum_{\hat{h}} exp(-E(v,\hat{h}))} \tag{3.48}$$

Niech:

$$F(v) = F_{P(h|v)}(v) \tag{3.49}$$

Wówczas pokazuje się, że:

$$exp(-F(v)) = \sum_h exp(-E(v,h)) \tag{3.50}$$

$$P(v) = \frac{exp(-F(v))}{\sum_{\hat{v}} exp(-F(\hat{v}))} \tag{3.51}$$

gdzie $\hat{v}$ indeksuje wszystkie możliwe stany warstwy widzialnej. Mianownik ma tu znaczenie normalizujące, tradycyjnie oznaczany jest przez $Z$.

Tym samym mamy:

$$P(a|s) = \frac{e^{-F(s,a)\cdot\beta}}{Z}. \tag{3.52}$$

Co oznacza, że jeśli tylko ujemna energia wolna par stan-akcja $-F(v) = -F(s,a)$ jest proporcjonalna do dyskontowanego wzmocnienia, maszyna Boltzmanna w stanie równowagi termodynamicznej realizuje strategię maksymalizującą wzmocnienie.

Jak znaleźć wagi tak, by minimalizacja energii (równoważna z minimalizacją energii wolnej) jednocześnie maksymalizowała zdyskontowane wzmocnienie?

Sallans i Hinton wychodzą ze spostrzeżenia, że:

$$\frac{\partial F(v)}{\partial w_{i,k}} = -v_i \langle h_k \rangle_{P(h_k|v)} \tag{3.53}$$

Proponują modyfikację wag w następujący sposób:

$$\Delta w_{i,k} \propto \Big(r_t + \gamma \hat{\mathcal{Q}}(s_{t+1}, a_{t+1}) - \hat{\mathcal{Q}}(s_t, a_t)\Big) v_i \langle h_k \rangle \tag{3.54}$$

gdzie $v = (s_t, a_t)$, a $\hat{\mathcal{Q}}(s,a) = -F(s,a)$.

Dla ograniczonej maszyny Boltzmanna dany jest jawny wzór na $F(s,a)$:

$$F(s,a) = F(v) = \; = -\sum_j \sum_i w_{i,j} v_i \langle h_j \rangle + + \sum_j \langle h_j \rangle \, log \, \langle h_j \rangle + (1 - \langle h_j \rangle) log(1 - \langle h_j \rangle). \tag{3.55}$$

Choć nie ma dowodu na zbieżność strategii realizowanej przez tak uczoną ograniczoną maszynę Boltzmanna, symulacja pokazuje, że metoda działa dobrze.

Hinton i Sallans badali swoją metodę na symulowanych częściowo obserwowalnych procesach decyzyjnych, w tym takich, które wymagały od agentów pracy zespołowej, a wzmocnienie było dawane po czasie - każdy z agentów działał mimo zerowego wzmocnienia, a cały zespół otrzymywał silne wzmocnienie gdy jeden z agentów osiągnął cel.

ROZDZIAŁ 4

# Projekt BrainLab

BrainLab to rozwijalna biblioteka programistyczna w języku Java umożliwiająca konstruowanie symulacji, w których mobilne bioty uczą się na podstawie eksploracji środowiska. Celem, jaki przyświecał mi przy projektowaniu biblioteki, było stworzenie systemu uczącego się w sposób podobny do organizmów żywych - nie w sensie zasady działania, ale metodologii zdobywania umiejętności.

## 4.1. Motywacja

Projekt BrainLab narodził się z rozważań na temat niezwykłej zdolności naturalnych sieci neuronowych do uczenia się. Każdy badany model systemu uczącego się lub sieci neuronowych jest zawsze uproszczeniem, wyidealizowaniem problemu. Funkcjonuje tu prawo - im bardziej szczegółowy model, im bardziej zbliżony do modelowanego oryginału, tym mniej jesteśmy w stanie na jego temat powiedzieć, obliczyć, udowodnić. Aby model był nie tylko eksperymentalny, ale też poddawał się obróbce matematycznej, trzeba zdecydować się na kompromis i zrezygnować z pewnych cech, czy możliwości oryginału. Spora część publikacji dotyczących sztucznych sieci na neuronach impulsujących ogranicza się do jednego impulsu na neuron, upraszcza funkcję ekscytacji do zwykłego sumowania lub bazuje wyłącznie na czasowym kodowaniu danych. Dlaczego? Ponieważ pełna implementacja sieci na neuronach impulsujących nie poddaje się łatwo żadnemu z aparatów matematycznych stosowanych w przypadku innych sieci - aktualizacja stanów (tu: wysyłanie impulsów) nie jest ani losowa, ani nie ma z góry określonej kolejności, czas refrakcji neuronu nie jest stały, a zależy od impulsów na wejściu i tak naprawdę nie możemy odgadnąć jaką postać będzie miał szereg impulsów na wyjściu, o ile nie uruchomimy modelu.

Częstym uproszczeniem przy modelowaniu systemów uczących się jest to, że zawsze występuje „nauczyciel”, „krytyk” lub inne zewnętrzne źródło danych, sugestii lub nagród[1]. Z wyjątkiem może szczurów laboratoryjnych i studentów[2], w przyrodzie nie obserwuje się zbiorów danych do nauczenia, czy regularnego treningu w identycznych warunkach, a

[1]W sieciach samoorganizujących się nie ma co prawda nauczyciela, ale wyniki uczenia wymagają interpretacji - sieć może podzielić dane na określoną ilość kategorii, ale dopiero „ręczna” analiza podziału pozwala te kategorie nazwać. Zwykle też, dane są znane, a od sieci oczekujemy tylko dobrej generalizacji.

[2]Dowcip laboratoryjny mówi, że student świetnie zastępuje szczura laboratoryjnego, w dodatku nie trzeba mu sprzątać klatki.

wolno żyjące szczury rozwiązują w swoim życiu nie mniej złożone problemy niż przejście przez labirynt w poszukiwaniu sera, czy odkrycie przycisku uwalniającego karmę[3]. Co sprawia, że Deep Blue gra w szachy porównywalnie z mistrzem świata, ale bez drastycznej przeróbki oprogramowania (po przeróbce to nie jest już ten sam Deep Blue) przegra w o wiele prostszą grę, jaką są warcaby? Czym jest ta wewnętrzna potrzeba dopasowania się, odkrycia metody, zapamiętania nowych informacji? Jak doszło do tego, że nasze specjalizowane przez tysiące lat kończyny potrafimy w ciągu jednego życia nauczyć pisać, malować, przeprowadzać operację na otwartym sercu, czy prowadzić pojazd? Dlaczego ośmiornica, umieszczona w nowym dla siebie środowisku - szklanym labiryncie, stara się ignorować mylący ją zmysł wzroku i zaczyna orientować się w przestrzeni wyłącznie za pomocą dotyku?

Bioty mają być właśnie uproszczonym modelem uczenia biologicznego, który nie otrzymuje danych w gotowych paczkach i nie ogranicza kierunku uczenia. Projekt ma charakter czysto badawczy (*proof of concept*), nie ma na celu praktycznych zastosowań, czy rozwiązywania w optymalny sposób trudnych problemów. Wszystko, czego mogą nauczyć się bioty, można zaimplementować prostym programem, z wyjątkiem samego uczenia się. Z założenia, biot ma działać jak wpuszczone w nieznane środowisko stworzenie, które odkrywa, co jest dla niego dobre. Problematyka inteligentnego agenta w nieznanym środowisku rozwiązywana jest przez uczenie ze wzmocnieniem, które można równie dobrze implementować za pomocą programowania dynamicznego, co sieciami neuronowymi. Zawsze jednak wymagany jest sygnał z zewnątrz - wózek balansujący odwrotne wahadło otrzymuje sygnał o kącie nachylenia wahadła i ciągłe uwagi o postępach. Agenci grający w piłkę nożną otrzymują informację o zdobytym przez jednego z graczy golu - nie widzą tego. Oczywiście, wymuszając samodzielność biotów w odkrywaniu świata, wymuszam też uproszczenie zmysłów biotów. W końcu można argumentować, że czujnik nachylenia wahadła jest zmysłem wózka i jest on samodzielnym bytem, próbującym wypełnić cel swojego życia, jakim jest utrzymywanie wahadła w pionie. Podobnie piłkarze, zajęci w innej części boiska, mogą nie widzieć momentu zdobycia gola, ale okrzyk kibiców jest dla nich wystarczającą informacją, że gol został zdobyty i zarazem wzmocnieniem, że dobrze spełnili swoją część zadania. Bioty mają symulować zachowania naturalne.

Wiele eksperymentów wskazuje na to, że ogromne znaczenie dla uczenia się nowych umiejętności ma zapamiętywanie wzorców. Osoby, które często piszą na klawiaturze, mają pewne słowa lub fragmenty słów „wprogramowane” w pamięć ruchową. Szczególnie wyraźne to jest u programistów, którzy powtarzają często te same słowa kluczowe. Programista często odruchowo wpisuje zapamiętane słowo, gdy słowo, które chciał wpisać

[3]W nowo wydanej książce, „The Math Instinct” Keith Devlin pokazuje przykłady obliczeń matematycznych wykonywanych przez zwierzęta.

rozpoczyna się tym samym ciągiem liter, np. „*private*ge" zamiast „privilege", „*void*ence" zamiast „violence". Wpisanie poprawnego słowa wymaga wysiłku umysłowego i trwa nieco dłużej - raz ułożone na klawiaturze dłonie „nie chcą" się zatrzymać w połowie słowa, by dokończyć je inaczej niż tak, jak by chciały.

Podobne zjawisko obserwuje się u szachistów, którzy zdają się inaczej reagować na konfiguracje figur i pionów na szachownicy, które są częścią gry, niż na takie, które zostały ułożone losowo - poproszeni o zapamiętanie i odtworzenie konfiguracji, mają o wiele lepsze wyniki w przypadku tych pierwszych. Dla porównania - laicy równie marnie odtwarzają konfiguracje naturalne dla gry, co losowe. Sugeruje to jakoby długoletnie doświadczenie w grze pozwoliło wyspecjalizować pamięć gracza w kierunku symbolicznych figur na biało-czarnej planszy. Skutek dla wydajności grania jest jasny - gracz, pamiętając układ z wcześniejszej gry, poświęca mniej czasu na analizę i jego wzrok szybciej pada na miejsce, które wcześniej uznał za strategicznie ważne lub natychmiast decyduje się wykonać ruch, który wcześniej doprowadził jego lub jego przeciwnika do zwycięstwa.

W przypadku zwierząt, ciężko o eksperyment, który by wykazał istnienie podobnego zjawiska. Można się jednak spodziewać, że występuje - zapamiętywanie wzorców w wewnętrznej reprezentacji jest procesem bardziej niskopoziomowym, prostszym niż generowanie strategii. Wiemy, że psy, koty, czy delfiny są w stanie zapamiętać nawet do kilkudziesięciu poleceń i opanować dość złożone sekwencje ruchów. Ciężko powiedzieć, czy u nich też występuje nieodparta chęć automatycznego dokończenia raz zaczętej sekwencji.

Idea uczenia się strategii przez twarde zapamiętywanie dużej ilości wzorców wydaje się kontrintuicyjna - programujemy nasze komputery tak, aby program miał małe wymagania pamięciowe, aby algorytm uwzględniał możliwie jak największą ilość przypadków. Ciężko sobie wyobrazić program, który uczy się swojego zadania przez powtarzanie prób i zapamiętywanie tych udanych. Jednak, jak już wspomniałem, komputer nie jest dobrym modelem mózgu naturalnego. Ten sposób uczenia się pozwala szybko (w sensie ilości prób) osiągać dobre wyniki w prostych zadaniach: aby upolować rybę za pomocą kawałka naostrzonego kija, wystarczy uderzyć w chwili, gdy ryba jest w odpowiednim miejscu. Prosta sekwencja - unieść rękę z kijem, odczekać, uderzyć, wyrzucić rybę na brzeg. Pierwsze próby na pewno są nieudane, jednak z każdą udaną próbą krystalizuje się technika, a ruchy stają się coraz szybsze, bardziej precyzyjne, wręcz automatyczne. Nasi przodkowie najprawdopodobniej nie rozważali skomplikowanej fizyki tego zadania - przesunięcia obrazu ryby w wodzie, szybkości uderzenia, z uwzględnieniem zmiany gęstości ośrodka w chwili uderzenia w lustro wody. Myślenie strategiczno-algorytmiczne przy każdej próbie z osobna jest mniej wydajne niż automatyczne uruchamianie odpowiedniej sekwencji w momencie, gdy stan otoczenia jest wystarczająco podobny do wcześniej zapamiętanego.

Fascynujące jest, że byliśmy w stanie przenieść tę metodologię na tak abstrakcyjne problemy jak szachy czy pisanie na klawiaturze.

Interakcja biota z otoczeniem w projekcie BrainLab zaprojektowana jest tak, by biot uczył się z doświadczenia. Tu pojawia się problem - we wszystkich wspomnianych wcześniej przypadkach zadanie jest dobrze sprecyzowane jeszcze przed wykształceniem jakiejkolwiek strategii, przed pierwszą próbą. Nowicjusz wie, że gra w szachy skończy się matem. Pisząc na klawiaturze planujemy napisać dane słowo. W każdym przypadku, mamy wewnętrzną reprezentację celu i choć szczątkową informację o metodzie przed podjęciem próby. Jakakolwiek specyfikacja zadania podsunięta uczącemu się automatowi jednocześnie sugeruje strategię: Dla poławiacza ryb specyfikacja ma postać instrukcji w znanym mu języku: ryba, ostry kij, woda, brzeg, do niego należy przetworzenie tej instrukcji na ruch. Jak reprezentować zadania dla prostego mózgu biota inaczej niż w postaci bezpośredniego przekazania pożądanej sekwencji ruchów? Jak oczekiwać od biota wypełnienia misji, jeśli nie przekazaliśmy mu co właściwie ma robić?

Pamięć autoasocjacyjna jest dość dobrym modelem pamięci istot żywych. Jak jednak zapewnić wybiórczość, która pozwala uczyć się z własnego doświadczenia? W praktycznych zastosowaniach sieci neuronowych zbiór danych do zapamiętania lub sklasyfikowania prawie zawsze jest dany. Potrzebna jest swoisty mechanizm uwagi, a to sprowadza nas do punktu wyjścia - aby z natłoku wektorów wybrać te interesujące, musimy wiedzieć, które są interesujące. Aby wiedzieć, które są interesujące, musimy wiedzieć jak wykonać zadanie, a to właśnie chcemy odkryć.

Zdaje się, że rolę takiego właśnie kontrolera uwagi spełnia u istot żywych złożony system emocji. Wiadomo, że wydarzenia związane z silnymi emocjami zapamiętujemy na dłużej i bardziej szczegółowo. Umiejętności, które przynoszą nam pozytywne doznania, przyswajamy sobie szybciej. Emocje i uczucia są w przypadku ludzi dość skomplikowane, ale u prostych istot żywych powodowane są przez określony zestaw bodźców oznaczających osiągnięcie chwilowego celu: znalezienie pożywienia, wody, czy zapewnienie przetrwania gatunku. W mózgu wyróżniono ośrodek nagrody i kary, a eksperymenty[4] potwierdziły jego znaczenie w procesie uczenia.

W świetle tych przemyśleń, zdecydowałem się na rozwiązanie rodem ze wspomnianej tresury zwierząt, minus polecenia. Większość szczeniaków w pierwszym dniu tresury nie zastanawia się najprawdopodobniej nad znaczeniem dźwięków wydawanych przez istotę ludzką. W naturze psa leży jednak dążenie do otrzymania nagrody - smakołyku, pogłaskania, na dalszym etapie interakcji z człowiekiem nawet pochwały przyjaznym tonem głosu (emocje, układ nagrody i kary). Raz nagrodzone, zwierzę stara się odtworzyć warunki,

[4]Szczur laboratoryjny z wszczepioną elektrodą stymulującą ośrodek nagrody do perfekcji opanowuje naciskanie przycisku przynoszącego mu przyjemność.

które przyniosły nagrodę i zaczyna kojarzyć bodziec (*siadł pies!*) z wykonaniem odpowiedniego ruchu lub sekwencji ruchów. Pierwszy sukces zazwyczaj jest przypadkowy, ale po pierwszym nagrodzeniu pies wykonuje polecenia coraz lepiej. Nie znaczy to bynajmniej, że wytresowany pies zna niuanse znaczeniowe pojęcia „siedzieć". Po prostu kojarzy wejście - komendę z wyjściem - czynnością ruchową. Dokładnie na tym polega elementarna funkcjonalność biotów.

Założenie o pierwszej, losowej próbie ograniczyło drastycznie pulę zadań, jakich mogą uczyć się bioty. Podobnie jak od szczeniaka nie możemy oczekiwać, by kiedykolwiek poprawnie wykonał komendę „graj w szachy!" czy choćby (co bardziej leży w granicach fizycznych psich umiejętności) - „wypij pół miski wody", od biotów nie możemy oczekiwać bardziej złożonego zachowania niż takie, które może się wykształcić wskutek stopniowego poprawiania sekwencji, do której biot doszedł przypadkiem.

Kolejne ograniczenie możliwości biotów jest skutkiem założenia o ich autonomiczności. Biot nie otrzymuje informacji o swojej bezwzględnej pozycji na planszy, o odległości wobec sąsiednich biotów czy innych obiektów w otoczeniu. Biot dysponuje jedynie tym, co można zaobserwować z jego wirtualnego punktu widzenia. W przyrodzie, problem punktu widzenia kompensuje się dzięki złożonym mechanizmom optyki odwrotnej (tworzenie wewnętrznej reprezentacji przestrzennej na podstawie dwóch płaskich, rozmytych obrazów) i po części - pamiętaniu wzorców (widząc coś, co przypomina nos, odtwarzamy trójwymiarowy kształt nosa). Od biotów nie można oczekiwać umiejętności tak złożonego przetwarzania informacji, choć pewna jego namiastka, w postaci wytwarzania wewnętrznej reprezentacji na podstawie dwuwymiarowej „optyki" jest konieczna do zadania, które bioty wykonują dość dobrze. Utrudnia to jednak zadania wymagające szczegółowej reprezentacji otoczenia, jak wspomniany w wcześniej ruch stadny (flocking). Bioty w najbliższym czasie nie będą grały w piłkę.

Mimo tak licznych ograniczeń, bioty uczą się, a dynamika ich ruchów jest dość realistyczna.

## 4.2. Gra

Wspomniana we wcześniejszym rozdziale biologicznie inspirowana metoda uczenia może być przedstawiona w schematyczny sposób jako seria „gier", czyli niezależnych podejść agenta do zadania, które mogą kończyć się sukcesem lub porażką. Podobnie jak tresowane zwierze jest w trakcie tresury poddawane powtarzającym się próbom, biot, eksplorując środowisko, raz na jakiś czas trafia na obiekt zainteresowania. Gdy zareaguje poprawnie i wykona zadanie, do jakiego został przeznaczony, otrzymuje impuls „emocjonalny", który powoduje, że zapamiętuje całą sekwencję par stan-akcja od początku gry, czyli od momentu

zauważenia obiektu. Jeśli zadanie nie zostanie wykonane, zapis sekwencji gry jest tracony i biot wraca do trybu eksploracji.

Aby więc nadać biotowi zadanie, należy ustalić warunek początkowy gry oraz warunek wygranej i przegranej, w języku aktywacji zmysłów. Tym zajmuje się klasa `Instinct`, o której później.

Impuls rozpoczynający grę powoduje, że wektory stan-akcja zaczynają być zbierane w tablicy modelującej pamięć krótkotrwałą. Pomyślne zakończenie gry powoduje utrwalenie zawartości tablicy w pamięci autoasocjacyjnej realizowanej przez maszynę Boltzmanna w mózgu biota. Dla pełnej biologiczności modelu, pamięć krótkotrwała powinna być siecią neuronową, ale brak dobrych rozwiązań umożliwiających utrwalenie lub odrzucenie zawartości neuronowej pamięci autoasocjacyjnej wymusił implementację tablicową.

Z punktu widzenia ograniczonej maszyny Boltzmanna w mózgu biota, raz na jakiś czas otrzymuje ona serię wzorców względnie mało odległych w przestrzeni możliwych stanów i akcji - kolejne pary stan-akcja stanowią punkty na pewnej krzywej w tej przestrzeni. Ze względu na skończoną pojemność sieci, nie można oczekiwać, by wzorce były dokładnie odtwarzane - szczególnie jeśli różnice między nimi są niewielkie. Zamiast tego, pamięć długotrwała biotów ma za zadanie utrwalić tą ścieżkę na krajobrazie energetycznym. Względna bliskość kolejnych wzorców sprawia, że ekstrapolacja pośrednich stanów jest łatwiejsza niż gdyby do pamięci krótkotrwałej trafiały tylko kluczowe momenty w wykonywaniu akcji. Praktyka pokazuje, że pojemność ograniczonej maszyny Boltzmanna nie pozwala na zapamiętywanie dużej ilości ścieżek i szybko następuje efekt zagubienia - z jednej strony sukces biota zależy od nauczenia się poprawnego reagowania w wielu możliwych sytuacjach, a z drugiej, różnorodność zapamiętanych sytuacji prowadzi do przepełnienia pamięci sieci i zachowania chaotycznego.

Rozważania na temat tego, co powinno się dziać w sytuacji, gdy biot nie wykona zadania poprawnie, doprowadziły do spostrzeżenia, że wobec szerokiej teorii o uczeniu maszynowym i uczeniu sieci neuronowych, brakuje dobrego przepisu na od-uczanie, czyli osłabianie wzorca.

Metoda uczenia maszyn Boltzmanna, ta nie-MCD, zawiera różnicę korelacji po danych (wzorcach) i korelacji po całej przestrzeni, w stanie stacjonarnym:

$$\Delta w_{ij} = \eta \cdot \left( \langle x_i x_j \rangle_{q^0} - \langle x_i x_j \rangle_{q^\infty} \right) \tag{4.1}$$

Narzuca się kilka rozwiązań problemu karania biota. Pierwszym jest uczenie go negatywu wzorca - w końcu biot, który ruszył w stronę niebezpieczeństwa, powinien się z tej porażki nauczyć ruszać w kierunku przeciwnym. Jednak maszyna Boltzmanna i tak zapamiętuje negatywy wzorców, a w gestii neuronu równoważącego jest pilnowanie, by negatywy nie były równoprawne z zapamiętanymi wzorcami. Uczenie negatywu miałoby więc sens tylko na poziomie wag między neuronem równoważącym, a pozostałymi.

Powstaje inny problem - oprócz akcji, wektory zawierają również stan. Nie interesuje nas uczenie biota, że gdy *NIE* widzi niebezpieczeństwa to powinien się wycofać. Należałoby więc odwrócić tylko część odpowiedzialną za akcję. To mogłoby odnieść pożądany skutek, ale rodzi kilka pytań. Czy rzeczywiście całą sekwencję ruchów należy odwrócić? Może tylko ostatnie ruchy były błędne? Co z efektorami antagonistycznymi? Należałoby więc nie tyle uczyć negatywu wzorców, co dla każdego efektora znajdować efektor antagonistyczny i odwracać aktywacje w parach. To jest pewną komplikacją i narzuca wymogi wobec architektury biota - aby każdy efektor miał efektor antagonistyczny. Tymczasem może się okazać, że lepszą strategią unikania niebezpieczeństwa jest użycie szybkiego efektora obracającego niż słabego „biegu wstecznego".

Innym rozwiązaniem może być uczenie biota wzorców zerowych. Reguła uczenia sprowadza się wtedy tylko do drugiego składnika, czyli średniej korelacji po przestrzeni danych w stanie stacjonarnym. Te, gdyby traktować jednostki ukryte jako widzialne, są na podstawie reguły Hebba proporcjonalne do samych wag. Uczenie wzorców zerowych oznacza więc odejmowanie od wag czegoś proporcjonalnego do nich - w końcu staramy się douczyć sieć tak, by dawała jako odpowiedź sygnał możliwie bliski zeru. Osłabiając wagi, osłabiamy reaktywność biota - będzie wykonywał powolniejsze ruchy, co może narazić go na kolejne porażki. Jednocześnie, przy niezmienionej stałej uczenia sprawiamy, że kolejny zapamiętany wzorzec (seria wzorców) będzie miał większe znaczenie niż wcześniej zapamiętane wzorce. Czy uzasadnione jest aby biot, który właśnie popełnił błąd, łatwiej przyjmował następną lekcję? Kolejne podejście do zadania może być poprawne, ale mniej optymalne.

Problemem przy karaniu biota jest to, że porażka nie daje informacji o postaci poprawnego rozwiązania. Aby wiedzieć, że od niebezpieczeństwa trzeba uciekać, potrzebna jest wysokopoziomowa wewnętrzna reprezentacja problemu - niemalże wyobraźnia. Najszybsza droga ucieczki nie zawsze jest odwróceniem najszybszej drogi podejścia do obiektu, a prosta obserwacja przyrody pokazuje, że czasem optymalną strategią wcale nie jest ucieczka, a na przykład atak, unik lub ukrycie się. U istot żywych unikaniem porażki zajmuje się często wbudowany system odruchów - stroszenia kolców, chowania się do skorupy, czy zmiany barwy. Zwierzę może najwyżej nauczyć się *nie* aktywować funkcji obronnych - oswoić się.

Wobec tych rozważań, zdecydowałem się na proste czyszczenie pamięci krótkotrwałej w razie porażki.

## 4.3. Symulowane środowisko

Symulacja w BrainLab składa się z prostokątnej planszy (klasa Ground) o topologii torusa i zbioru umieszczonych na niej obiektów: biotów, ścian i pożywienia. Wszystkie zamieszczone na planszy obiekty rozszerzają klasę Prop (z ang. rekwizyt), która przechowuje położenie obiektu. Bioty (klasa Agent) rozszerzają Prop, ale mają w symulacji szczególne względy.

Każdy ruchomy obiekt (a więc przede wszystkim bioty) porusza się na zasadzie napędu, tarcia i bezwładności. Czas symulacji jest dyskretny, a model musi uwzględniać siły pochodzące z efektorów biotów. W tym celu zaprojektowałem prosty model fizyki w dwóch wymiarach.

Obiekt ruchomy ma w każdym kroku przypisany wektor przesunięcia i skalarną wartość oznaczającą kąt obrotu (o ile ma możliwość się obracać).

Zderzenia w BrainLab są niesprężyste. Dla systemu wykrywania kolizji, okrągły obiekt jest lokalnym polem wektorów siły skierowanych od jego środka na zewnątrz. Im bliżej środka obiektu, tym silniejsze odpychanie. Nie ma zachowania momentu pędu - zderzenie dwóch biotów ma wpływ tylko na kierunek ich ruchu, nie na obrót.

W każdym kroku symulacji, dla każdego obiektu rozszerzającego Prop (o ile został on umieszczony w symulacji) sprawdzane jest, czy nie koliduje on z innymi obiektami rozszerzającymi tę klasę. W praktyce uwzględniane są jedynie zderzenia biotów ze ścianami, z pożywieniem (tu następuje zjedzenie, nie odbicie) i z innymi biotami.

Nową symulację tworzy się przez rozszerzenie klasy Ground (teren):

```
\label{ground}
brainlab.Ground g = new brainlab.Ground(400,400);
g.setBirthplace(15,15,150,150);
```

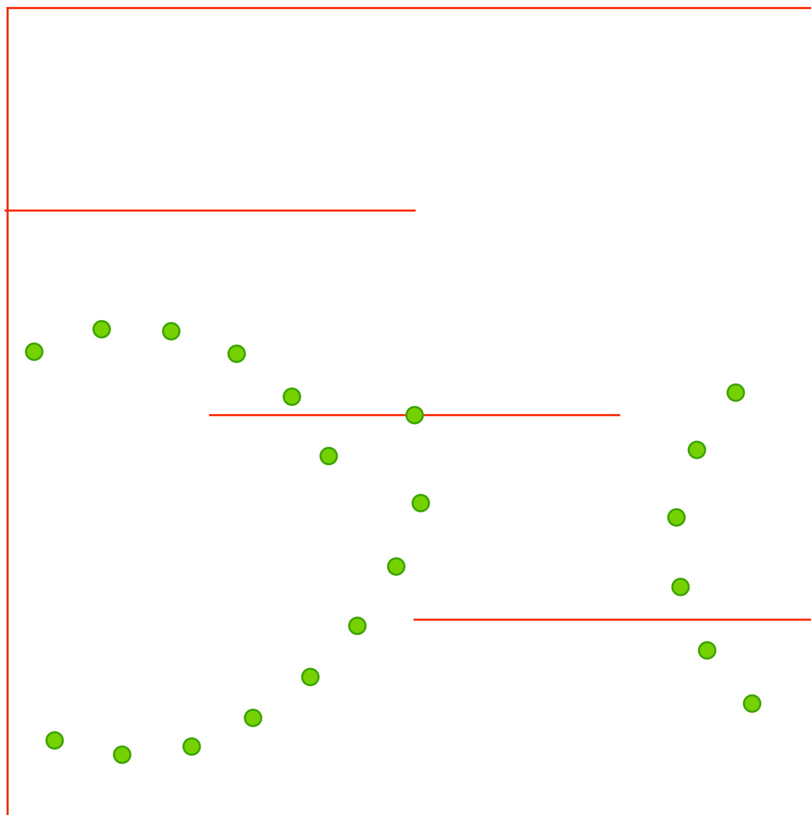
Tu teren ma rozmiar 400 na 400 pikseli, a bioty będą się pojawiały wewnątrz kwadratowego obszaru w pobliżu lewego-górnego rogu.

Następnie wypełnia się planszę nieruchomymi obiektami:

```
Wall W1= new Wall(false,1,1,398);
    Wall W2= new Wall(true,1,1,398);
    Wall W3= new Wall(true,1,399,398);
    Wall W4= new Wall(false,399,1,398);
    Wall W5= new Wall(true,0,100,200);
    Wall W6= new Wall(true,100,200,200);
    Wall W7= new Wall(true,200,300,200);
g.addProp(W1);
```

```
    g.addProp(W2);
    g.addProp(W3);
    g.addProp(W4);
    g.addProp(W5);
    g.addProp(W6);
    g.addProp(W7);
    g.addProp(new Food(200,200));
    for(int i=0; i<20; i++){
     g.addProp(new Food(10,10,380,380));
}
```

Parametry konstruktora klasy `Wall` określają czy ściana ma być wschodnio-zachodnia, czy północno-południowa (parametr typu `boolean`), współrzędne północno-wschodniego końca i długość. Porcja pożywienia zainicjowana dwiema współrzędnymi pojawi się w miejscu wyznaczonym przez te współrzędne. Gdy podane są cztery współrzędne, porcja pojawi się wewnątrz prostokąta, w pobliżu ostatnio dodanej do planszy porcji pożywienia.



**Rysunek 4.1**. Teren zawierający pożywienie i ściany.

Zanim przejdziemy do dodania biotów, przyjrzyjmy się strukturze biota.

## 4.4. Anatomia biota

Założenie o dostarczaniu biotowi tylko tych informacji o symulowanym świecie, jakie jest w stanie sam zdobyć, jest odwzorowane w anatomii biota. Wizualny „awatar" biota to kolorowe kółko z „czułkami", których jedynym znaczeniem jest pokazywanie, gdzie jest przednia część biota - nie mają żadnego związku z rzeczywistymi sensorami czy motorami. Taka prosta forma odzwierciedla to, że zadaniem biotów jest uczyć się - model nie przewiduje rozwoju fizycznego jednostki, modularności ciała biota, czy efektorów zmieniających jego kształt (choć to ostatnie może być na dłuższą metę interesujące). Biot jest z założenia organizmem jednokomórkowym.

W wizualnej reprezentacji znajdować się może zasięg sensorów środowiskowych, reprezentowany jako przezroczyste okręgi.

Klasa `Anatomy` zawiera metodę rysującą biota, ale przede wszystkim odpowiada za ilość, typy, parametry i układ sensorów i efektorów.

Jeśli biot ma mieć anatomię, która nie była wcześniej zdefiniowana (anatomia klasy `Anatomy` nie zawiera żadnych sensorów ani efektorów), musimy rozszerzyć klasę `Anatomy`. Nazwijmy nową klasę `Beetle` i nadajmy jej metodę rysującą żukowaty kształt[5]:

```
// ... odpowiednie deklaracje

public class Beetle extends Anatomy{

public void draw(Graphics2D graphic, int X, int Y, Color color, double rotation, double sca
graphic.setColor(color);
graphic.fillOval(
    (int)Math.round((X-radius)*scale),
(int)Math.round((Y-radius)*scale),
(int)Math.round(2*radius*scale),
(int)Math.round(2*radius*scale));
graphic.setColor(new Color(0,0,0));
graphic.drawOval(
    (int)Math.round((X-radius)*scale),
    (int)Math.round((Y-radius)*scale),
    (int)Math.round(2*radius*scale),
    (int)Math.round(2*radius*scale));
graphic.drawLine(
```

[5]Taki żukowaty kształt jest domyślny dla klasy `Anatomy`, więc fragment kodu ma tylko znaczenie informacyjne.

```
    (int)Math.round(X+scale*radius*Math.sin(rotation-0.5)),
    (int)Math.round(Y-scale*radius*Math.cos(rotation-0.5)),
    (int)Math.round(X+scale*(radius+5)*Math.sin(rotation-0.5)),
    (int)Math.round(Y-scale*(radius+5)*Math.cos(rotation-0.5)) );
graphic.drawLine(
    (int)Math.round(X+scale*radius*Math.sin(rotation+0.5)),
    (int)Math.round(Y-scale*radius*Math.cos(rotation+0.5)),
    (int)Math.round(X+scale*(radius+5)*Math.sin(rotation+0.5)),
    (int)Math.round(Y-scale*(radius+5)*Math.cos(rotation+0.5)) );
}
// ... c.d.n.
```

### 4.4.1. Sensory

Sensory lub zmysły to abstrakcyjne obiekty, powiązane z neuronami wejściowymi(stanu) wewnętrznej sieci neuronowej biota. Sensorowi przypisana jest ilość neuronów, z którymi się kontaktuje i funkcja obliczająca jego aktualną aktywację. Najprostszym przykładem sensora jest taki, który wysyła do przypisanego sobie neuronu stałą wartość. Celem istnienia sensora o stałej, niezerowej wartości jest zmniejszanie prawdopodobieństwa wystąpienia negatywu wzorca na etapie przywoływania.

Sensory kontekstowe lub opóźnione to takie, których aktywacja równa jest aktywacji wybranych neuronów z poprzedniego kroku. Można w ten sposób zwiększać wymiarowość wejścia (zwykle stan sprzed jednego kroku nie różni się znacznie od obecnego), jak też dawać biotom pewną ograniczoną pamięć krótkotrwałą, umożliwiającą naukę reakcji nie tylko na stan obecny, ale na zmiany stanu pomiędzy krokami. Biotom wyposażonym w taką duplikację zmysłów zdarzało się ponownie podchodzić do straconego z zasięgu celu.

Wymienione wyżej rodzaje sensorów określam mianem *sensorów wewnętrznych* dla odróżnienia od klasy *sensorów środowiskowych*.

Sensory środowiskowe przyjmują swój stan na podstawie stanu otoczenia. Sensor środowiskowy ma wyznaczony zasięg (pole widzenia) i rozkład czułości wewnątrz swojego zasięgu. W implementacji jest to koło o środku w pobliżu ciała biota, a siła reakcji na obiekt w zasięgu zależy liniowo od odległości obiektu od środka pola widzenia:

$$act = \begin{cases} \frac{r+\rho-d(obj)}{r+\rho} & d(obj) < r+\rho \\ 0 & w.p.w. \end{cases} \tag{4.2}$$

gdzie $\rho$ to promień obiektu (np. innego biota), $r$ to zasięg sensora, a $d(\ldots)$ to odległość środka obiektu od środka pola widzenia sensora.

Oprogramowanie przewiduje również obiekty o innym kształcie niż kolisty (np. podłużne ściany). Odległość w takim przypadku liczona jest odpowiednio:

```
public double dist(double X, double Y){
        if(isHorizontal()){
         if((X>getX())&&(X<getX()+getLength()))
return Math.abs(Y-getY());
            if(X<getX())
             return Functions.distance(X,Y,getX(),getY());
            if(X>getX()+getLength())
             return Functions.distance(X,Y,getX()+getLength(),getY());
        }else{
            if((Y>getY())&&(Y<getY()+getLength()))
             return Math.abs(X-getX());
            if(Y<getY())
             return Functions.distance(X,Y,getX(),getY());
            if(Y>getY()+getLength())
             return Functions.distance(X,Y,getX(),getY()+getLength());
        }
        return -1; // dla spokoju sumienia
}
```

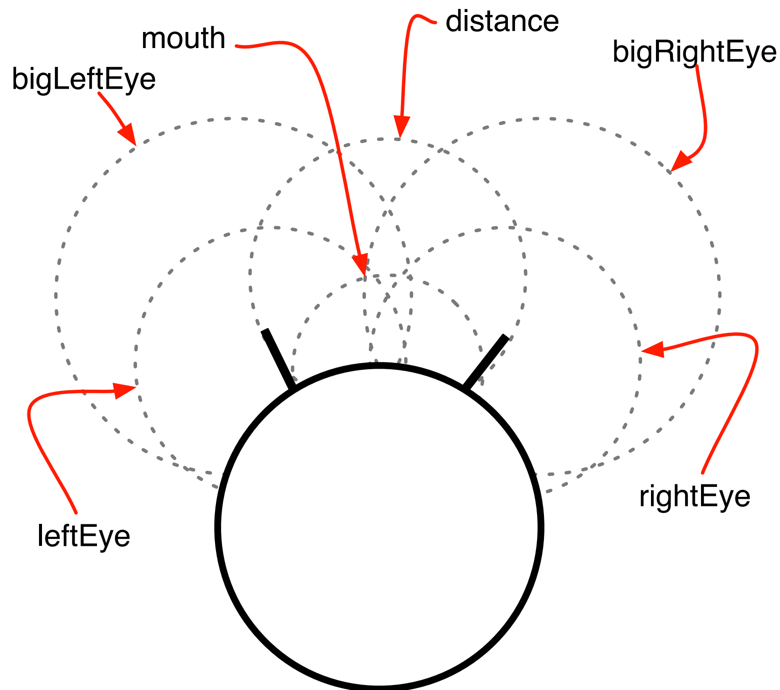
Metoda `distance` klasy `Functions` realizuje odległość euklidesową na płaszczyźnie.

Dodajmy do klasy `Beetle` zestaw sensorów:

```
//... c.d. klasy Beetle

public EnvironmentSensor getEatingSensor(){
return mouth;
}

public Beetle(){
leftEye = new EnvironmentSensor(1,15,0.5,14);
rightEye = new EnvironmentSensor(1,15,-0.5,14);
bigLeftEye = new EnvironmentSensor(1,20,0.6,20);
bigRightEye = new EnvironmentSensor(1,20,-0.6,20);
distance = new EnvironmentSensor(1,15,0,15);
mouth = new EnvironmentSensor(2,5,0,5);
addSensor(leftEye);
```




**Rysunek 4.2**. Rozkład zasięgu sensorów biota z przykładu.

```
addSensor(rightEye);
addSensor(bigLeftEye);
addSensor(bigRightEye);
addSensor(distance);
addSensor(mouth);
addSensor(new ConstantSensor(1.0));

bigLeftEye.canSenseFood(true);
bigRightEye.canSenseFood(true);
leftEye.canSenseFood(true);
rightEye.canSenseFood(true);
mouth.canSenseFood(true);
distance.canSenseFood(true);
// ... c.d.n.
```

Parametry konstruktora sensorów środowiskowych to ilość neuronów połączonych z sensorem, promień zasięgu i współrzędne biegunowe położenia względem środka biota: kąt i promień. Ostatni dodany sensor to sensor stały. Sensor „paszczowy" (`mouth`) został

wyznaczony jako sensor jedzący - pożywienie będzie musiało się znaleźć w jego zasięgu, by mogło być zjedzone.

W tym fragmencie kodu należy zakładać, że wszystkie sensory zostały wcześniej zadeklarowane. Aby dany sensor reagował na ściany lub inne bioty, wystarczy dodać:

```
bigLeftEye.canSenseWalls(true);
bigLeftEye.canSenseAgents(true);
```

Sensor środowiskowy może reagować na wszystkie te klasy obiektów lub tylko na wybrane. W przypadku stworzenia nowej klasy obiektów, należy też stworzyć nowy typ sensora - rozszerzyć klasę `EnvironmentSensor`, aby reagowała na nowe obiekty.

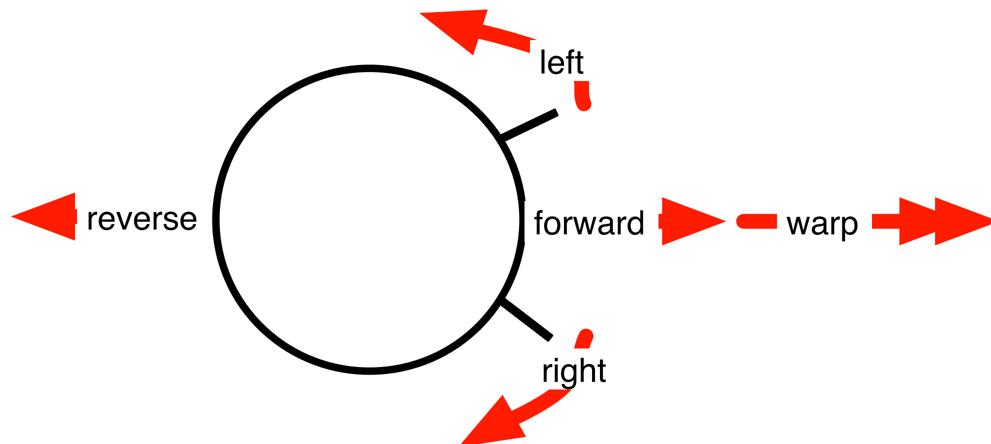
### 4.4.2. Efektory

Biot porusza się dzięki efektorom. W implementacji, klasa efektora (klasa `Motor`) obejmuje każdy obiekt zdolny pobierać aktywację z przypisanego sobie neuronu. Głównym zadaniem efektorów jest umożliwianie biotom poruszania się, ale efektor nie musi powodować ruchu biota - może np. modyfikować parametr uczenia lub powodować efekt wizualny.

Efektory ruchowe występują w dwóch postaciach - napędzające i obracające biota. [6] Efektor napędzający ma przypisaną *siłę*, która determinuje odległość, o jaką przesunięty może zostać biot pod wpływem tego efektora. Analogicznie, efektor służący biotowi do obracania się ma przypisany kąt o jaki w jednym skoku obraca się biot. Rzeczywista siła (lub moment siły) wywołana przez aktywację efektora jest iloczynem siły efektora i stopnia aktywacji powiązanego z nim neuronu, o ile aktywacja neuronu przekroczyła charakterystyczną dla każdego efektora wartość progową. Jeśli siła aktywacji neuronu nie przekracza wartości progowej, aktywacja efektora nie następuje. Chociaż neurony w modelu są bipolarne, domyślną wartością progową jest zero i zakłada się, że wartość progowa poniżej zera jest niedopuszczalna - ujemna aktywacja neuronu powodowałaby ujemną aktywację efektora. Na wzór biologicznych mięśni każdy efektor ma odpowiadać za konkretny ruch. Ruch w przeciwnym kierunku może być realizowany przez antagonistyczny wobec niego efektor.

Zestaw efektorów biota, podobnie jak zestaw sensorów, powinien być dopasowany do zadania, jakie biot ma wykonać. Biot, który musi poruszać się szybko, powinien mieć efektory umożliwiające dalekie skoki. Biot, którego zadanie wymaga precyzyjnego sterowania, powinien być wyposażony w serię efektorów o małej sile. System wartości progowych pozwala projektować efektory o dużej sile, ale aktywowane tylko w szczególnych sytuacjach - gdy biot jest „pewien”, że nagły, daleki skok jest opłacalny - aktywuje efektor o wysokim progu aktywacji i wykonuje szybki ruch naprzód.

[6] We wcześniejszej fazie projektu jeden efektor mógł jednocześnie obracać i poruszać biotem.




**Rysunek 4.3**. Efektory biota według fragmentu kodu, szkic.

Dajmy biotowi dwa efektory poruszające go do przodu - jeden słabszy, drugi silniejszy, słaby „bieg wsteczny” oraz dwa antagonistyczne efektory do obrotu:

```
//...c.d. Beetle
forward = new Thrust(0.2,6);
warp = new Thrust(0.7,10);
reverse = new Thrust(0.0,-2);
left = new Turn(0.0,-0.2);
right = new Turn(0.0,0.2);
addMotor(forward);
addMotor(warp);
addMotor(reverse);
addMotor(left);
addMotor(right);
}

} // koniec klasy Beetle.
```

Parametry kreatora efektora to próg (minimalna aktywacja neuronu wymagana do aktywacji efektora) i siła, mierzona w pikselach lub w radianach odpowiednio.

## 4.5. Instynkt

Ponieważ BrainLab uwzględnia możliwość współistnienia biotów wykonujących różne zadania w tej samej symulacji, zaistniała potrzeba zdecentralizowania implementacji kryty-

ka, czyniąc z niego indywidualny moduł każdego biota. Z punktu widzenia uczenia ze wzmocnieniem, instynkt nie jest częścią agenta (tu - biota), tylko częścią środowiska.

Instynkt to prosty system wartościowania działania biota, jedyny element jego architektury, który „wie" cokolwiek na temat zadania, jakie biot ma spełniać. Na wzór instynktu istoty żywej, który reaguje na głód, niebezpieczeństwo, ale też zadowolenie czy sytość, instynkt biota reguluje proces uczenia przez wysyłanie sygnałów o rozpoczęciu zadania i jego pozytywnym lub negatywnym zakończeniu do mózgu.

Instynkt jest dla biota drugą, obok mózgu, jednostką kontrolującą jego zachowanie. W przeciwieństwie do mózgu, nie jest to jednostka neuronowa, nie podlega uczeniu i jest ustalana dla danego biota (klasy biotów) na początku jego życia i pozostaje niezmieniona. Aby spełnić zasadę braku podpowiedzi, instynkt z założenia otrzymuje częściową tylko informację ze zmysłów: otrzymuje sygnał, że jedzenie znalazło się w zasięgu zmysłu, ale nie którego (ta informacja dociera tylko do mózgu). Rola instynktu została niebezpośrednio omówiona na początku tego rozdziału - klasa `Instinct` jest symulacją uwagi i emocji, selektorem wzorców, które powinny znaleźć się w zbiorze uczącym.

W implementacji, instynkt jest podstawowym elementem odróżniającym bioty różnych gatunków - określa ich zadanie. Biot przeznaczony do szukania porcji pożywienia, będzie otrzymywał od instynktu wzmocnienie w momencie zjedzenia porcji (`onEat`).

Załóżmy, że nasz biot będzie szukał jedzenia. Klasa `EaterInstinct` rozszerza klasę `Instinct`:

```
private class EaterInstinct extends Instinct{

public void onFindFood(){
getAgent().getBrain().startGame();
}

public void onEat(){
getAgent().getBrain().winGame();
}

public void onLoseFood(){
getAgent().getBrain().loseGame();
}
}
```

Nasz biot nie ma jeszcze mózgu.

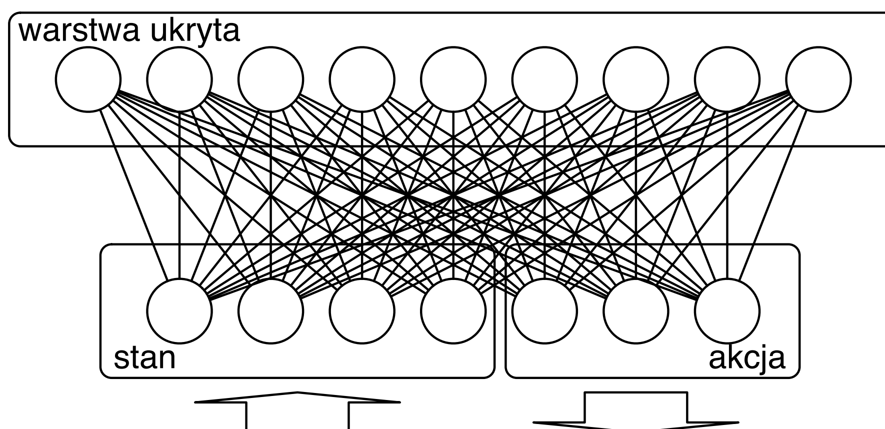
## 4.6. Mózg

Środowisko BrainLab nie stawia ograniczeń na postać mózgu biota. Klasa `Brain` i klasy z nią powiązane zakładają jedynie, że mózg będzie przyjmował i zwracał sygnały w postaci wektorów liczb z przedziału $[0, 1]$. Nie narzuca to wymagania, by sama wewnętrzna sieć neuronowa operowała na domenie ciągłej - sygnał może być przez obiekt klasy `Brain` przetwarzany na postać zero-jedynkową przez ustalenie progu i w takiej postaci przetwarzany.

W bardzo wczesnej fazie projektu, mózg biotów były dyskretnymi maszynami Boltzmanna, bez jednostek ukrytych, uczonymi regułą Hebba. Wtedy właśnie narodził się pomysł pół-krokowej fizyki świata - aby wygładzić skokowy ruch biotów. Model ciągły był jednak potrzebny, by wprowadzić sensory o ciągłej aktywacji. Następnym krokiem było wykorzystanie reguły MCD do uczenia.

Zanim mózgi biotów były ograniczonymi maszynami Boltzmanna, przeprowadziłem eksperyment używając do uczenia algorytmu będącego swoistą mieszanką uczenia hebbowskiego i MCD. Synapsy wewnątrz warstwy widzialnej uczone były regułą Hebba, a synapsy między warstwą widzialną a ukrytą - uproszczonym uczeniem MCD: korelacje były zliczane ze stanu sieci po krótkiej relaksacji (nie podczas relaksacji) i nie podlegały uczeniu współczynniki pochyłości funkcji sigmoidalnych. W efekcie dawało się zauważyć asymetryczne zachowanie biotów - częściej używany z pary sensorów lub efektorów bardzo szybko zaczynał dominować i biot wędrował w kółko, ignorując jedzenie.

W finalnej wersji, mózg jest ograniczoną maszyną Boltzmanna na domenie ciągłej, uczoną za pomocą reguły minimalizacji zbieżności kontrastującej. W tej wersji nie pojawia się efekt krążenia.




**Rysunek 4.4**. Schemat mózgu biota

Zadaniem mózgu jest dopasowanie akcji do stanu - autoasocjacja wektora, w którym część zawierająca opis zaobserwowanego stanu jest dana, a część zawierająca akcję jest niepełna. W każdym kroku animacji wagi sieci decydują o następnym ruchu biota na podstawie aktywacji sensorów.

Sfinalizujmy klasę naszego biota, nadając mu wcześniej stworzony instynkt, anatomię i mózg oparty na ciągłej, ograniczonej maszynie Boltzmanna:

```
// ... pomijam oczywiste deklaracje

public class Eater extends Agent {

public Eater(String newName){
super(newName);
setInstinct(new EaterInstinct());
setAnatomy(new Beetle());
setBrain(new CRBMBrain(getAnatomy(), 15));
setEatingSensor(((EaterAnatomy)myAnatomy).getEatingSensor());
}
}
```

Konstruktor klasy `CRBMBrain` przyjmuje jako parametry obiekt klasy `Anatomy` (lub klasy potomnej) i rozmiar warstwy ukrytej. Anatomia jest potrzebna przy budowaniu mózgu - wyznacza rozmiar warstwy widzialnej. Każdy neuron wejściowy jest połączony z jakimś sensorem, każdy wyjściowy - z efektorem.

## 4.7. Dynamika biota

Bioty poruszane są przez program asynchronicznie, w kolejności, w jakiej były dodane do symulacji. Każdy biot wykonuje swój ruch, a następne, przy ustalaniu aktywacji zmysłów, obserwują jego nową pozycję. Dla obserwatora, nie jest to zauważalne - ruch biotów jest płynny (z wyjątkiem momentów uczenia się, gdy rozmiar sieci neuronowej jest duży).

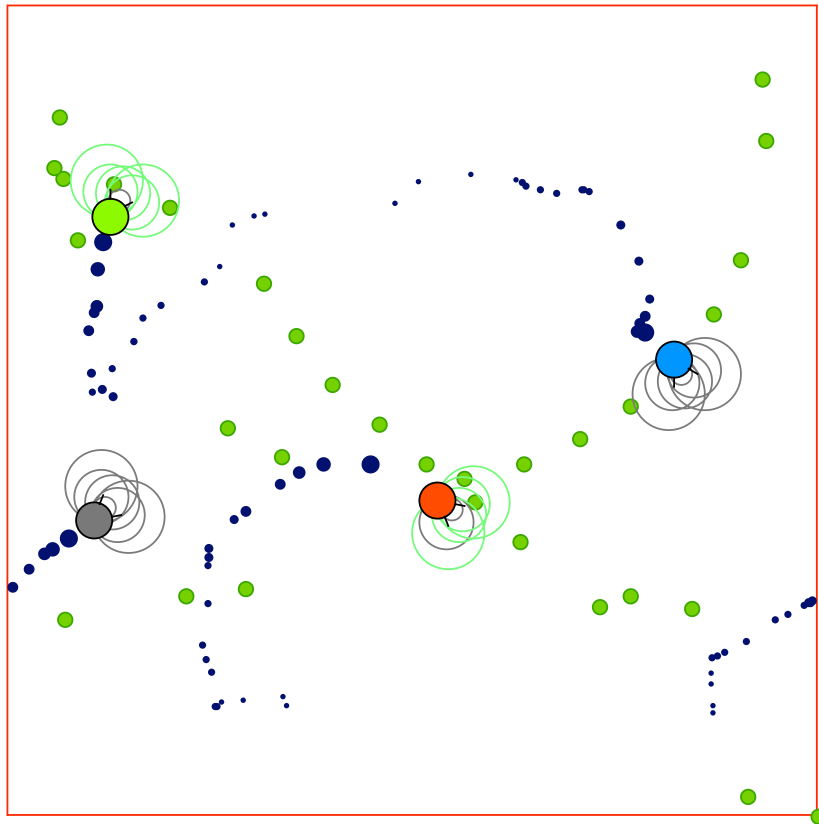
Każda klatka animacji biotów składa się z następujących etapów:

1. Uaktualnianie stanu zmysłów

   Zmysły środowiskowe są uaktualniane na podstawie stanu otoczenia, zmysły wewnętrzne obliczają swoją aktywację.

2. Instynkt

   Mając do dyspozycji aktywację zmysłów, instynkt modyfikuje parametry mózgu, aktywuje lub deaktywuje tryb uczenia.



**Rysunek 4.5**. Scena z symulacji.

3. Aktywność mózgu

   Aktywacje zmysłów stają się aktywacjami neuronów wejściowych. Następuje pojedynczy cykl relaksacji i nowe aktywacje neuronów wyjściowych stają się aktywacjami efektorów. Na tym etapie nie jest jeszcze wykonywany ruch.

4. Wykrywanie kolizji

   Do wektora siły dodawane są wektory wynikające z wykrytych kolizji między biotem, a innymi biotami, czy ścianami. Nadal nie jest wykonywany rzeczywisty ruch biota.

5. Aktywacja efektorów

   Przesunięcie wynikające z aktywacji efektorów dodawane jest do wektora siły.

6. Ruch i zamknięcie cyklu

   Biot przesuwa się o połowę wektora siły (tarcie) i wektor siły jest połowiony do wykorzystania w następnym kroku biota.

W kodzie klasy `Simulation` ta kolejność odzwierciedla się następująco:

```
    public void step(){
        //Acquisition of sensory input
        myGround.feedSensors();
        //Basic instincts
        myGround.activateAgentInstincts();
        //Cerebral activity
        myGround.thinkAgents();
        //Detecting collisions
        myGround.collideProps();
        myGround.collideAgents();
        //Effectors
        myGround.activateMotors();
        //Finishing the cycle
        myGround.moveAgents();
    }
```

## 4.8. Eksperyment

Bioty zostały poddane testowi - na planszy umieszczony został biot i agent o identycznej anatomii, o identycznym instynkcie, ale z mózgiem zastąpionym generatorem liczb losowych z zakresu, w jakim mogłaby działać maszyna Boltzmanna.

Dodajmy do naszej przykładowej symulacji trzy bioty uczące się i biota „głupiego”:

```
Eater w1 = new Eater("Eater blue  ");
w1.setColor(new Color(0,192,255));
g.addProp(w1);

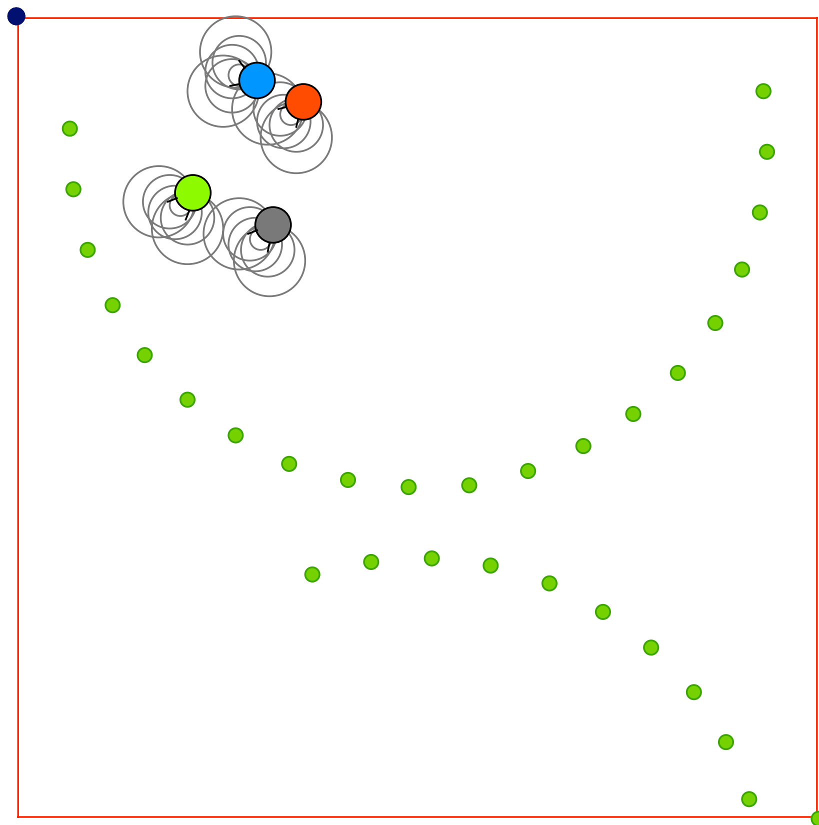
Eater w2 = new Eater("Eater orange");
w2.setColor(new Color(255,153,0));
g.addProp(w2);

Eater w3 = new Eater("Eater green ");
w3.setColor(new Color(0,255,192));
g.addProp(w3);

Eater wan = new Eater("Wanderer       ");
wan.setBrain(new StupidBrain(wan.getAnatomy(), 0));
```

```
wan.setColor(new Color(102,102,102));
g.addProp(wan);

Simulation sim = new Simulation(g);
```



**Rysunek 4.6**. Cztery bioty na wspólnym terenie, początek symulacji.

Gdzie g to wcześniej (str. **??**) stworzony obiekt klasy `Ground`.

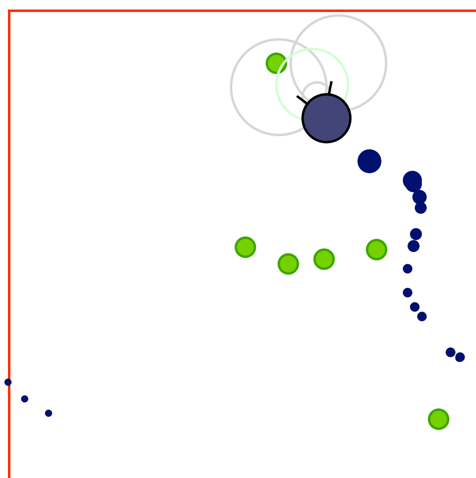
Tu warto zauważyć, że `Wanderer` to `Eater`, czyli biot jedzący, ale jego mózg jest klasy `StupidBrain`.

Budowę symulacji kończy stworzenie okna zawierającego nową symulację:

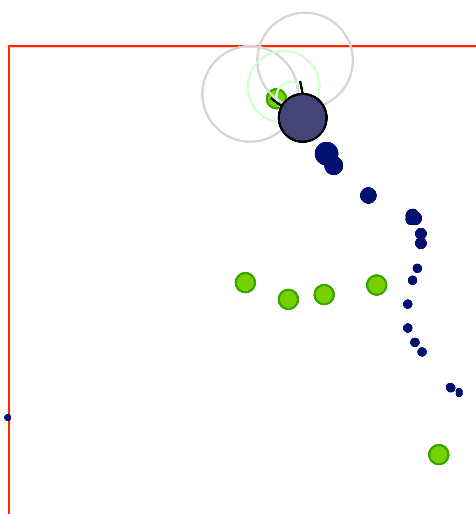
```
new SimulationFrame(sim);
```

Wyniki biotów mierzone były przez proporcję „gier" wygranych do przegranych, jak też przez sumaryczną ilość gier wygranych - w tym przypadku porcji pożywienia zjedzonego przez danego biota.

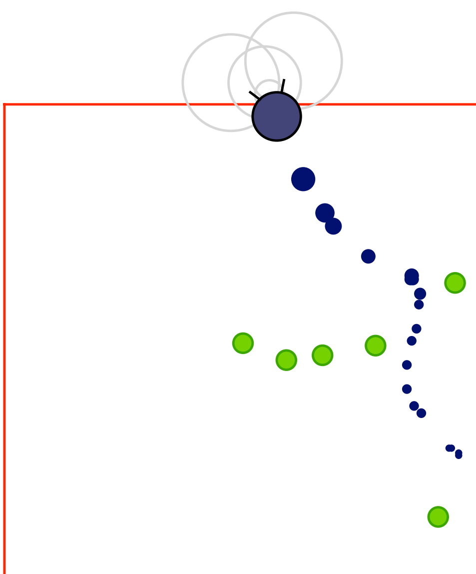
Ta miara efektywności biota w żadnym wypadku nie jest miarą doskonałą. Gdy biot zobaczy dwie porcje pożywienia i zdecyduje się na jedną, druga będzie policzona jako



**Rysunek 4.7**. Biot podchodzi do jedzenia...



**Rysunek 4.8**. ...celuje...



**Rysunek 4.9**. ...zjada porcję i uczy się na bazie ostatniego doświadczenia.

porażka. Za porażkę policzona zostanie również porcja, która znalazła się na krawędzi zasięgu zmysłów. Bioty mają tendencję do rozglądania się, więc porcja na krawędzi zasięgu zmysłów może być policzona wielokrotnie[7]. Czasem daje się zaobserwować, że biot wykonał poprawny ruch w kierunku pożywienia, ale pod wpływem nagłego impulsu (stochastyczność sieci) nagle odskoczył lub cofnął się[8]. Na wynik liczbowy wydajności biota wpływa więc zasięg zmysłów, gęstość rozstawienia pożywienia, konfiguracja efektorów i ogólne parametry mózgu, dlatego skupię się na porównaniu do biota poruszającego się losowo.

W symulacji, po efektach brzegowych na samym początku, daje się obserwować powolny wzrost wydajności biotów uczących się (do 35 procent przy parametrach na jakich testowałem) i w początkowej fazie wyraźne zróżnicowanie umiejętności poszczególnych biotów uczących się. Wydajność biota losowego w tej fazie powoli spada do typowego dla niego (przy tych parametrach symulacji) poziomu 15-20 procent. W miarę, gdy wydajność biotów uczących się wyrównuje się (około 50 pomyślnych „gier”), zaczyna spadać do 20-25 procent, co prawdopodobnie spowodowane jest przepełnieniem pamięciowym sieci. Na tym etapie daje się zauważyć dysproporcja w *ilości* pomyślnych prób w czasie między biotem losowym, a uczącymi się. Ilość porcji zjedzonych przez biota losowego jest zawsze w okolicach połowy ilości porcji zjedzonych przez bioty uczące się (średnio). Ta tendencja jest zachowana nawet w bardzo długich symulacjach. Wyniki po kilkuminutowej symulacji[9]:

```
Eater green      w/l: .23        eats: 169
Eater orange     w/l: .20        eats: 135
Eater blue       w/l: .20        eats: 127
Wanderer         w/l: .19        eats: 74
```

Jak wyjaśnić fenomen, że mimo porównywalnej wydajności, bioty uczące się mają na koncie więcej sukcesów?

We wcześniejszej fazie rozwoju projektu, w podobnej symulacji biot losowy uzyskiwał gorszy wynik w wydajności, ale zjadał więcej pożywienia. Było to spowodowane najprawdopodobniej mniejszą niż obecnie używana warstwą ukrytą, większą temperaturą w sieci i większym zagęszczeniem pożywienia. Biot losowy atakował pożywienie „na oślep”, a biot uczący się popełniał błędy w najprostszych nawet sytuacjach. Tu sytuacja odwraca się. Bioty uczące się poruszają się szybciej i dzięki skalibrowanym parametrom uczenia mają bardziej precyzyjne ruchy. Daje się zaobserwować, że rozglądają się, gdy rzeczywiście są w pobliżu pożywienia i zawsze zjadają porcję, która znajduje się bezpośrednio przed

[7] Jedna porcja pożywienia może więc przynieść wiele przegranych gier, ale tylko jedną wygraną.

[8] Za liczenie tego za porażkę ciężko winić miarę efektywności - to *jest* porażka biota.

[9] Losowej, nie wybranej jako reprezentacyjny przypadek

nimi. Bioty uczące się mają więcej sukcesów przy porównywalnej wydajności, ponieważ dokonują większej ilości prób.

Zadanie polegające na omijaniu ścian (wszystkie zmysły ustawiamy na wykrywanie ścian, odpowiednio modyfikujemy instynkt, aby utrata ściany z pola widzenia była wygraną, a zderzenie - przegraną) kończy się fiaskiem - choć chwilami widać postępy, szybko wydajność biotów uczących się i losowego wyrównuje się.

ROZDZIAŁ 5

# Perspektywy rozwoju

Projekt BrainLab oparty jest na dalekosiężnym planie uzupełnienia rozlicznych symulacji typu sztucznego życia o zdolność poszczególnych agentów do uczenia się w sposób, w jaki uczą się istoty żyjące. Mimo sukcesów w prostych zadaniach, projektu nie można uznać za ostateczne wypełnienie planu. Wręcz przeciwnie - możliwych dróg rozwoju projektu jest wiele, zarówno w kwestii architektury biota, jego mózgu, sensorów i efektorów, jak i w kwestii nowych zadań do nauczenia, miar skuteczności w wykonaniu zadań i nowych możliwości, jakie można dać biotom.

Dalszy rozwój może wymagać modyfikacji restrykcyjnych reguł, aby możliwe było wypełnianie trudniejszych zadań, jednak wielką wagę przykładam do tego, by bioty pozostały bytami autonomicznymi. Podobnie jak boidy, których grupowe zachowanie wynikało z reguł rządzących jednostkami, złożone zachowania kolonii biotów mają być zachowaniami wynikającymi z ich indywidualnej umiejętności uczenia się.

## 5.1. Sieci oparte na neuronach impulsujących

We 1952 roku Alan Lloyd Hodgkin i Andrew Fielding Huxley przeprowadzili serię eksperymentów, których efektem było pięć publikacji tworzących precyzyjny opis elektrochemiczny aksonu neuronu biologicznego. Obiektem ich badań był neuron kałamarnicy olbrzymiej, dostatecznie duży, by można było do niego podłączyć elektrody, a wynikiem - równania opisujące zależności między poziomem potasu, sodu, a opornością neuronu. Na podstawie serii karkołomnych pomiarów, byli w stanie stworzyć schemat układu elektrycznego realizującego funkcjonalność neuronu kałamarnicy. Hodgkin, Huxley i Australijczyk, Sir John Carew Eccles w 1963 odebrali za ten model nagrodę Nobla z dziedziny medycyny.

Wszystkie stosowane modele sieci neuronowych mniej lub bardziej czerpią ich wniosków - od idei sumowania wartości na wejściu, poprzez stosowanie sigmoidalnych funkcji przejścia, po asynchroniczne uaktualnianie stanu. Obecnie pojawiają się liczne publikacje dotyczące matematycznego modelu opartego na równaniach Hodgkina i Huxleya.

Sztuczne neurony impulsujące to złożony model, obejmujący funkcjonalnością inne modele sieci neuronowych i daleko wykraczający możliwościami poza wcześniej proponowane sieci, co wzbudza nadzieję, że dobrze zbadany, pozwoli nam z łatwością konstruować

jednolite sieci, które będą jednocześnie odszumiały, klasteryzowały i klasyfikowały, uczone bez nadzoru[1].

Kolejność aktywacji w sieciach impulsujących wyznaczana jest przez same neurony, czas jest ciągły, dopuszcza się wiele synaps o różnych opóźnieniach między dwoma neuronami, a refrakcja nie ma określonego czasu trwania.

Przez impuls rozumie się deltę Diraca skupioną w określonej chwili $t_i$, impuls jest identyfikowany z momentem w czasie, w którym został wysłany.

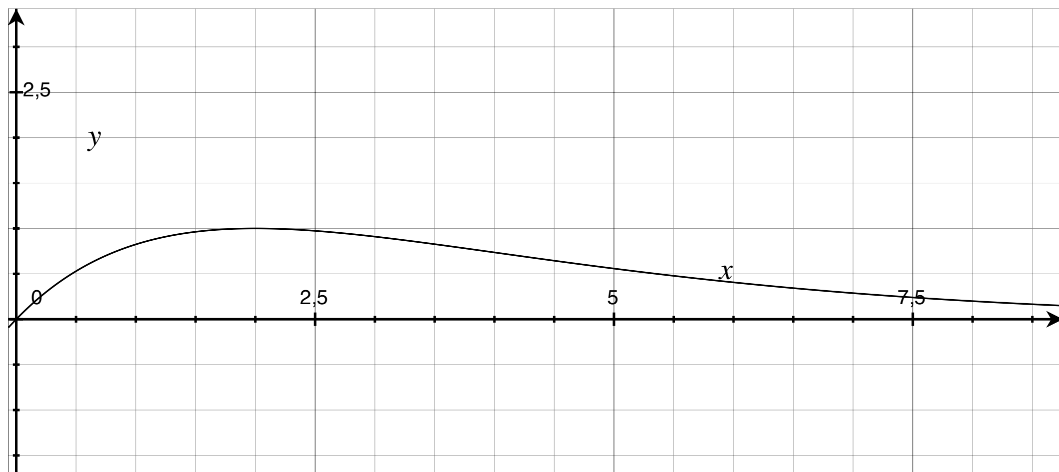
Załóżmy, że $i$-ty neuron wysłał impuls do neuronu $j$-ego w chwili $t_i$. W chwili $t$, neuron $j$-y

$$y_i^k(t) = \epsilon(t - t_i - d^k) \tag{5.1}$$

gdzie $t$ to ciągły czas, $t_i$ to chwila nadejścia impulsu, $d^k$ to opóźnienie na $k$-tej synapsie, a $\epsilon(t)$ to funkcja określająca wpływ impulsu w czasie:

$$\epsilon(t) = \frac{t}{T} e^{1-\frac{t}{T}}, \tag{5.2}$$

parametr $T$ odpowiada za dodatkowe opóźnienie reakcji, ale też wydłuża wpływ.




**Rysunek 5.1**. Funkcja $\epsilon$ - wpływu impulsu na neuron w sieciach impulsujących

Z wykresu 5.1 widać, że nadchodzące impulsy są przez neuron „zapominane” z czasem.

Wynikowa aktywacja neuronu $j$ zależna jest również od wag na synapsach:

$$x_j(t) = \sum_{i \in N_j} \sum_{k=1}^{m} w_{i,j}^k y_i^k(t) + \eta(t - \hat{t}_j) \tag{5.3}$$

gdzie $N_j$ to zbiór sąsiadów neuronu $j$, a $\eta(t)$ to funkcja wycieku aktywacji, zależna od czasu ostatniego impulsu tegoż neuronu: $\hat{t}_j$. Funkcja $\eta$ powinna być silnie ujemna dla wartości bliskich zeru i powoli rosnąć do zera - odpowiada za refrakcję.

[1] To zdanie wyraża mieszankę nadziei i sarkazmu i jako takie powinno być odczytane.

Wobec braku wpływających impulsów, aktywacja neuronu zbiega do zera. Aby wzrosła, neuron musi otrzymać impuls lub serię impulsów od synaptycznych sąsiadów. Gdy aktywacja $x_j(t)$ przekracza pewną arbitralnie ustaloną graniczną wartość, neuron wysyła impuls do wszystkich swoich następców. Przyjmuje się, że siła impulsów jest stała (regulowana jedynie przez wagi), a minimalna wartość aktywacji wymagana do nadania impulsu jest porównywalna z siłą impulsu.

Aby neuron „wystrzelił", musi albo otrzymać szybką serię impulsów, albo silny impuls od sąsiada, z którym łączy go synapsa o dużej wadze. Czas refrakcji regulowany jest przez nadchodzące impulsy - jeśli są one w stanie pokonać hamujące działanie funkcji $\eta(t)$, neuron wystrzeli szybciej. Po dłuższym czasie, do pobudzenia neuronu może wystarczyć jeden nadchodzący impuls.

W implementacji, ciągły czas zastępuje się kolejką priorytetową, do której dodawane są momenty nadawania i odbierania impulsów przez kolejne neurony. W ten sposób, o ile znamy czasy impulsów na wejściu sieci, całe dalsze zachowanie sieci jest w pełni zdeterminowane, ale niezwykle trudne do przewidzenia.

Model obliczeń realizowany przez sztuczne sieci impulsujące odbiega od tego we wcześniejszych modelach. Ponieważ naturalnym językiem neuronów impulsujących są właśnie impulsy, potrzebne są nowe reprezentacje danych. Powszechnie stosuje się dwie.

W reprezentacji cyfrowej, zero-jedynkowa informacja jest taktowana - co regularny odcinek czasy następuje impuls jeśli była jedynka lub nic nie następuje jeśli było zero. Parametr $T$ w 5.2 reguluje położenie maksimum funkcji $\epsilon(t)$, więc czas, po którym neuron najsilniej reaguje na impuls. Ta charakterystyczna opóźniona reakcja sprawia, że neurony synchronizują się we wspólny rytm.

W analogowej, *gęstościowej* reprezentacji danych, impulsy następują tym częściej, im silniejszy był sygnał wejściowy, podobnie jak wczesne mikrofony membranowe, które dźwięk przekształcały w serię impulsów elektrycznych. Ta reprezentacja danych wydaje się być bardziej umotywowana biologicznie. Dla opartych o neurony impulsujące biotów, zmysł środowiskowy działałby jak licznik Geigera - im silniejsza aktywacja, tym częściej pojawiałyby się impulsy.

W rzeczywistości, oprócz nowych metod reprezentowania danych wejściowych i, co trudniejsze, interpretowania wyjścia neuronów, dziedzina sztucznych sieci impulsujących czeka na nowe algorytmy uczenia, w pełni wykorzystujące możliwości tych neuronów. Obecnie stosuje się w symulacjach kilka niepełnych modeli, ograniczonych tak, by upodobnić sieć impulsującą do innej. Prawdziwym wyzwaniem jest opracowanie metody uczenia modelu pełnego.

W przypadku biotów, chciałoby się zastosować uczenie regułą Hebba, ale wobec tak skomplikowanych zależności czasowych zachodzących między neuronami, zarówno na

poziomie synaps, jak i opóźnionych aktywacji, nie sposób przyjąć wysłania przez dwa neurony impulsu w tym samym momencie za wystarczający powód do zwiększenia wagi między nimi. Przede wszystkim - wagi której z synaps łączących te dwa neurony? Wszystkie są wykorzystywane równie często, choć w różnych momentach czasu.

## 5.2. Modularyzacja mózgu

Sieci impulsujące występujące w naturze mają skłonność do wewnętrznej synchronizacji całych grup neuronów, a badania pokazują, że takie zsynchronizowane grupy specjalizują się w jednej dziedzinie, np. w analizie sygnałów z jednego zmysłu, czy sygnałów o określonym charakterze.

Jakkolwiek ideałem byłoby, aby monolityczna sieć podłączona do sensorów i efektorów biota podzieliła się na wyspecjalizowane komórki „obsługujące" poszczególne funkcje biota, może istnieć konieczność odgórnego zaprojektowania architektury mózgu spełniającej określone zadanie. Przy wspólnym sygnale o spełnieniu zadania, wszystkie części inicjowałyby algorytm uczenia dla swoich danych wejściowych i wyjściowych, które by specyfikowały funkcję tych części. Układ wykrywający jedzenie mógłby wtedy ciągnąć biota w stronę jedzenia, układ wykrywający niebezpieczeństwo włączałby efektor cofający, a względna siła ich aktywacji decydowałaby o aktywacji odpowiednich efektorów.

Teoretycznie, taka sytuacja możliwa jest w maszynie Boltzmanna, która stanowi iloczyn ekspertów, jednak jej ograniczoność, choć drastycznie przyspiesza obliczenia, jednocześnie zmniejsza możliwości obliczeniowe - neurony ukryte są w zasadzie tylko perceptronami, niezdolnymi do dłuższych ścieżek wnioskowania.

## 5.3. Algorytmy genetyczne

Myśląc o biotach, staram się myśleć o modelu sztucznego życia, w którym jednostki doskonalą się same, a nie wskutek rekombinacji ich genów. Bioty mają być z założenia autonomiczne, dopasowywać się do nowych sytuacji i indywidualnie odkrywać swoje przeznaczenie.

Praktyka pokazuje, że nie jest to tak łatwe jak mogłoby się zdawać. Wydajność biota zależy nie tylko od czasu uczenia, ale od stanu środowiska i parametrów samego biota. Nawet w tak prostym eksperymencie jak nauka jedzenia, wydajność biota ściśle zależy od konfiguracji sensorów i efektorów. Co więcej - inna konfiguracja jest optymalna gdy jedzenie jest gęsto rozłożone, inna gdy średnia odległość między porcjami jest większa niż zasięg zmysłów biota. W procesie uczenia, ustawienie zbyt niskiej temperatury znacznie spowalnia proces uczenia (patrz: maszyny Boltzmanna a sieci Hopfielda), ustawienie temperatury zbyt wysokiej przyspiesza ten proces, ale też czyni biota bardziej chaotycznym.

Ewentualny schemat wyżarzania lub *tempering*, zwiększa tylko ilość parametrów do kalibracji. Proste uzależnienie temperatury od chwilowej wydajności nie poprawia wydajności. Do wymagających kalibracji parametrów dochodzi również rozmiar warstwy ukrytej i stała uczenia.

Zastosowanie algorytmu genetycznego do rozwoju biotów rozwiązałoby ten problem. Każdy biot zaczynałby życie w stanie „dziecinnym", w którym musiałby nauczyć się tego, czego już się nauczyły starsze bioty. Jednak jego genotyp nosiłby informacje o jego anatomii, temperamencie, skłonności do przyjmowania nowej wiedzy. Miarą przystosowania genotypu byłaby długość życia, ilość zjedzonych porcji pokarmu w ciągu życia lub na jednostkę czasu.

Propozycja zastosowania algorytmów genetycznych do biotów wymaga szczegółowego planowania.

ROZDZIAŁ 6

# Zakończenie

Pisząc tę pracę chciałem przekazać szerokie tło historyczne projektu, jednocześnie przedstawiając wykorzystaną w projekcie teorię. Starałem się, by opisy poszczególnych gałęzi wiedzy zawierały nie tylko ścieżkę prowadzącą do elementów wykorzystanych w BrainLab, ale były uzupełnione o teoretyczne, futurystyczne, a nawet filozoficzne rozważania. Dziedzina sztucznego życia, jak każda dziedzina interdyscyplinarna, wymaga szerokiego spojrzenia na problem i ciągłego pogłębiania wiedzy z dziedzin pokrewnych.

Symulacja a-life, jaką po części jest BrainLab, zawsze musi być kompromisem między elementami czerpanymi z biologii, a możliwością implementacji w postaci programu komputerowego i nawet tak rozwinięty projekt jak Framsticks, który wchodzi właśnie w etap uwzględniania chemii symulowanego świata, pozostaje ograniczony przez pierwotne założenia projektu - aby wszystko dało się zapisać w postaci kodu genetycznego. Framsticki doskonalą się przez genetykę, ale ich „inteligencja” sprowadza się do złożenia emiterów i filtrów sygnału analogowego, sterującego mięśniami - framsticki nie mają stanów wewnętrznych i choć potrafią reagować na otoczenie za pomocą prostych zmysłów, nie są w stanie się uczyć. Podobnie bioty, których *podstawowym* zadaniem jest uczyć się, mogłyby działać lepiej i wykonywać szerszą gamę zadań, gdyby ich sieci neuronowe miały pewne wagi ustalone na bazie ewolucji lub część ich zachowań była realizowana przez algorytm.

Założenie o braku podpowiedzi miało na celu sprawdzenie, czy możliwa jest implementacja uczenia, w którym agent widzi środowisko tylko przez swoje zmysły i sam dla siebie jest krytykiem. Eksperyment uważam za udany - bioty uczące się znajdują pożywienie lepiej i szybciej niż bioty poruszające się w sposób losowy. Choć zdarza im się zachowywać irracjonalnie i gubić poszczególne porcje, nawet krótka obserwacja pokazuje, że dynamika ich ruchu jest bardziej złożona niż byłaby, gdyby rządził nimi algorytm: bioty wykazują indywidualne tendencje, niektóre lubią atakować pożywienie, inne uczą się wracać do porcji, którą dopiero minęły. Przypadkowy obserwator, nie wprowadzony w tematykę minimalizacji rozbieżności kontrastywnej w ograniczonych maszynach Boltzmanna celem rozwiązywania częściowo obserwowalnych, wektoryzowalnych procesów decyzyjnych Markowa, natychmiast personifikuje bioty, przypisując im więcej inteligencji, niż rzeczywiście posiadają.

Znajomość zasad działania biotów ujmuje trochę tej magii i wprowadza uczucie niedosytu - w końcu one tylko zapamiętują korelacje między sensorami a efektorami, a ich zdolność rozumowania jest równie ograniczona, co ich wewnętrzne maszyny Boltzmanna

z dokładnie jednym neuronem ukrytym między każdym sensorem, a efektorem. Przypomina się krążący po Massachusetts Institute of Technology koan[1] opowiadający o dwóch legendach katedry sztucznej inteligencji:

W czasach, gdy Sussman[2] był jeszcze studentem, Minsky[3] podszedł do niego, gdy ten hakował[4] coś przy PDP-6[5].

– Co pan robi? - spytał Minsky

– Uczę sieć z losowymi połączeniami[6] grać w Kółko i Krzyżyk - odpowiedział Sussman.

– Dlaczego ta sieć ma losowe połączenia? - spytał Minsky

– Bo nie chcę, żeby miała jakieś sugestie, jak grać. - powiedział Sussman

Minsky wtedy zamknął oczy.

– Dlaczego zamyka pan oczy? - spytał Sussman swojego mistrza.

– Bo chcę, żeby pokój był pusty.

W tym momencie, Sussman został oświecony.

Tak naprawdę w każdej sformalizowanej symulacji zawarte są wskazówki co do jej celu. Bioty *byłyby* losowe, gdyby nie instynkt, zaprogramowany na uruchamianie procesu uczenia w odpowiedniej chwili. Podobnie, nie można wyobrazić sobie, by bioty same wynalazły algorytm uczenia, który utrwala wcześniej zaobserwowane wzorce. To samo można powiedzieć o boidach, które s, o grze Life, w której wszystkie skończone konfiguracje można ponumerować liczbami naturalnymi i rozrysować między nimi graf, w pełni determinujący zachowanie nawet tak złożonych konfiguracji jak maszyna Turinga[7]. Zachowania wynikające, które obserwujemy u prostych systemów zawsze mają wyjaśnienie, leżące w samej naturze tych systemów.

To rozczarowujące stwierdzenie stawia nas przed wielkim pytaniem - jeśli w zachowaniach sztucznych zwierząt nie ma nic niezwykłego, a tylko matematyka, to jak to jest z prawdziwymi zwierzętami? Czy ich zachowania też można wyjaśnić za pomocą dużych liczb naturalnych lub równań różniczkowych? A jak to jest z nami?

[1]Krótka przypowieść lub zagadka używana w Buddyzmie Zen celem pokazania błędów w rozumowaniu i oświecenia słuchacza.

[2]Gerald Jay Sussman, profesor z MIT.

[3]Marvin Minsky, legenda sztucznej inteligencji.

[4]W żargonie MIT, *hacker* to honorowy tytuł.

[5]Programmed Data Processor-6, model komputera firmy DEC. Sprzedano 23 sztuki, głównie uniwersytetom.

[6]Sieć o losowych połączeniach to wynalazek Minsky'ego.

[7]To przy założeniu posiadania ogromnej kartki papieru i zapasu ołówków.

DODATEK A

# Program i prezentacja na CD

# Bibliografia

[1] S. M. Bohte *Spiking Neural Networks* praca doktorska, marzec 2003

[2] H. Chen, A. F. Murray, *A Continuous Restricted Boltzmann Machine with a Hardware-Amenable Learning Algorithm* International Conference on Artificial Neural Networks 358?363, 2002

[3] H. Chen, A. F. Murray, *A Continuous Restricted Boltzmann Machine with an Implementable Training Algorithm* IEE Proc. of Vision, Image, Signal Processing, in press (2003)

[4] P. Cichosz, *Systemy uczące się* Wydawnictwa Naukowo Techniczne, Warszawa 2000

[5] M. Gardner *MATHEMATICAL GAMES The fantastic combinations of John Conway's new solitaire game "life"* Scientific American 223 (Październik 1970): 120-123.

[6] G.E. Hinton, *Training Products of Experts by Minimizing Contrastive Divergence* Neural Computation 14(8) 1771?1800, 2002

[7] J. Korbicz, A. Obuchowicz, D. Uciński *Sztuczne Sieci Neuronowe, Podstawy i Zastosowania* Akademicka Oficyna Wydawnicza PLJ, Warszawa 1994

[8] M. J. Mataric. *Interaction and Intelligent Behavior* PhD thesis, MIT EECS, 1994.

[9] T. Mitchell, *Machine Learning*, McGraw Hill, 1997.

[10] Z. Michalewicz, *Genetic Algorithms Data Structures = Evolution Programs* Springer 1992

[11] *Advanced Automation for Space Missions* Proceedings of the 1980 NASA/ASEE Summer Study June 23-August 29, 1980 NASA Conference Publication 2255

[12] S. Osowski, *Sieci neuronowe w ujęciu algorytmicznym* Wydawnictwa Naukowo-Techniczne, Warszawa 1996

[13] T. Quick, K. Dautenhahn, C. L. Nehaniv, G. Roberts, *On Bots and Bacteria: Ontology Independent Embodiment* Proceedings of ECAL 1999

[14] C. W. Reynolds. *Flocks, herds, and schools: A distributed behaviroal model* Computer Graphics, str 25?34, 1987.

[15] G. Rummery, M. Niranjan, *On-Line Q-Learning Using Connectionist Systems* Technical Report, Cambridge University Engineering Department, 1994

[16] D. Rutkowska, M. Piliński, L. Rutkowski, *Sieci neuronowe, algorytmy genetyczne i systemy rozmyte* Wydawnictwo Naukowe PWN

[17] Brian Sallans, Geoffrey W. Hinton, *Reinforcement Learning with Factored States and Actions* Journal of Machine Learning Research 5 (2004) 1063?1088 Submitted 3/02; Revised 1/04; Published 8/04

[18] Brian Sallans, *Learning Factored Representations for Partially Observable Markov Decision Processes* S. A. Solla, T. K. Leen, and K.-R. Muller, editors, Advances in Neural Information Processing Systems, volume 12. The MIT Press, Cambridge, 2000.

[19] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction* MIT Press, Cambridge, MA, 1998.

[20] M. Szurek, *Opowieści matematyczne* Wydawnictwa Szkolne i Pedagogiczne, Warszawa 1987

[21] C. Watkins, *Learning from Delayed Rewards* Thesis, University of Cambidge, England, 1989

# Spis rysunków

# Skorowidz